PB80-190986

Efficient Algorithms for Solving Systems of Ordinary Differential Equations for Ecosystem Modeling

State Univ. of New York Coll. at Plattsburgh

Prepared for

Environmental Research Lab.
Athens, GA

Mar 80

U.S. Department of Commerce
National Technical Information Service
NTIS

TECHNICAL REPORT DATA
(Please read Instructions on the reverse before completing)

1. REPORT NO.
EPA-600/3-80-037

2.

3. RECIPIENT'S ACCESSION NO.
PB 80 190986

4. TITLE AND SUBTITLE
Efficient Algorithms for Solving Systems of Ordinary Differential Equations for Ecosystem Modeling

5. REPORT DATE
March 1980 issuing date

6. PERFORMING ORGANIZATION CODE

7. AUTHOR(S)
John Malanchuk, John Otis, and Hubert Bouver

8. PERFORMING ORGANIZATION REPORT NO.

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Institute for Man and Environment
State University of New York
Plattsburgh, New York 12901

10. PROGRAM ELEMENT NO.
A28B1A

11. CONTRACT/GRANT NO.
R805452

12. SPONSORING AGENCY NAME AND ADDRESS
Environmental Research Laboratory--Athens GA
Office of Research and Development
U.S. Environmental Protection Agency
Athens, Georgia 30605

13. TYPE OF REPORT AND PERIOD COVERED
Final, 7/77-8/78

14. SPONSORING AGENCY CODE
EPA/600/01

15. SUPPLEMENTARY NOTES

16. ABSTRACT

This report presents three packages of subroutines for the numerical solution of systems of first order ordinary differential equations. The three integration methods were chosen to provide an efficient means of solving the full range of initial value problems encountered in basin ecosystem modeling. The subroutines have been designed to handle all aspects of the numerical integration as automatically as possible so that their successful use does not require a sophisticated knowledge of the techniques of numerical analysis.

After a general introduction to the principles and concepts of numerical integration, the particular methods used in the codes are discussed in some detail. Full instructions for use are given, and some simple examples are presented. The complete code listings, in ANSI FORTRAN IV, are contained in appendices.

17. KEY WORDS AND DOCUMENT ANALYSIS

a. DESCRIPTORS	b. IDENTIFIERS/OPEN ENDED TERMS	c. COSATI Field/Group
Systems analysis Mathematical models Environments		62B 68D 72E

18. DISTRIBUTION STATEMENT
RELEASE TO PUBLIC

19. SECURITY CLASS *(This Report)*
UNCLASSIFIED

20. SECURITY CLASS *(This page)*
UNCLASSIFIED

21. NO. OF PAGES

22. PRICE

EPA Form 2220-1 (9-73)

iA

★ U.S. GOVERNMENT PRINTING OFFICE: 1980-657-146/5637

United States
Environmental Protection
Agency

Environmental Research
Laboratory
Athens GA 30605

EPA-600/3-80-037
March 1980

Research and Development

PB80-190986

EPA

Efficient Algorithms for Solving Systems of Ordinary Differential Equations for Ecosystem Modeling

RESEARCH REPORTING SERIES

Research reports of the Office of Research and Development, U.S. Environmental Protection Agency, have been grouped into nine series. These nine broad categories were established to facilitate further development and application of environmental technology. Elimination of traditional grouping was consciously planned to foster technology transfer and a maximum interface in related fields. The nine series are:

1. Environmental Health Effects Research
2. Environmental Protection Technology
3. Ecological Research
4. Environmental Monitoring
5. Socioeconomic Environmental Studies
6. Scientific and Technical Assessment Reports (STAR)
7. Interagency Energy-Environment Research and Development
8. "Special" Reports
9. Miscellaneous Reports

This report has been assigned to the ECOLOGICAL RESEARCH series. This series describes research on the effects of pollution on humans, plant and animal species, and materials. Problems are assessed for their long- and short-term influences. Investigations include formation, transport, and pathway studies to determine the fate of pollutants and their effects. This work provides the technical basis for setting standards to minimize undesirable changes in living organisms in the aquatic, terrestrial, and atmospheric environments.

This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.

EPA-600/3-80-037
March 1980

EFFICIENT ALGORITHMS FOR SOLVING SYSTEMS OF ORDINARY DIFFERENTIAL EQUATIONS FOR ECOSYSTEM MODELING

by

John Malanchuk, John Otis and Hubert Bouver*

Institute for Man and Environment
State University of New York
Plattsburgh, New York 12901

*Applied Physics Laboratory
Johns Hopkins University
Laurel, Maryland 20810

Grant No. R805452

Project Officer

Ray R. Lassiter
Environmental Systems Branch
Environmental Research Laboratory
Athens, Georgia 30605

ENVIRONMENTAL RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
ATHENS, GEORGIA 30605

i8

DISCLAIMER

This report has been reviewed by the Athens Environmental Research Laboratory, U.S. Environmental Protection Agency, and approved for publication. Approval does not signify that the contents necessarily reflect the views and policies of the U.S. Environmental Protection Agency, nor does mention of trade names or commercial products constitute endorsement or recommendation for use.

FOREWORD

Environmental protection efforts are increasingly directed towards preventing adverse health and ecological effects associated with specific compounds of natural or human origin. As part of this Laboratory's research on the occurrence, movement, transformation, impact, and control of environmental contaminants, the Environmental Systems Branch studies complexes of environmental processes that control the transport, transformation, degradation, and impact of pollutants or other materials in soil and water, assesses environmental factors that affect water quality, and develops mathematical models for evaluating the environmental impact of pollutants.

Numerical integration packages for mathematical modeling, although widely available, are usually documented in the language of the numerical analyst. The unsophisticated user often has difficulty with the terminology and frequently chooses a routine that is far from optimal. This report provides useful information and comprehensible guidance to one particular group of users--mathematical modelers with backgrounds in subject areas such as biology. The objective is to provide easily accessible routines capable of efficiently addressing the class of problems often encountered by these modelers. The documentation is extensive and rigorous yet couched in explicit terms that are easily understood. Although nothing new appears here from the point of view of a numerical analyst, it is hoped that modelers will find the attempt at lucidity to have been worthwhile to them.

David W. Duttweiler
Director
Environmental Research Laboratory
Athens, Georgia

ABSTRACT

This report presents three packages of subroutines for the numerical solution of systems of first order ordinary differential equations. The three integration methods were chosen to provide an efficient means of solving the full range of initial value problems encountered in basin ecosystem modeling. The subroutines have been designed to handle all aspects of the numerical integration as automatically as possible, so that their successful use does not require a sophisticated knowledge of the techniques of numerical analysis.

After a general introduction to the principles and concepts of numerical integration, the particular methods used in the codes are discussed in some detail. Full instructions for use are given, and some simple examples presented. The complete code listings, in ANSI FORTRAN IV, are contained in Appendices.

This report was submitted in fulfillment of Grant No. R-805452 by Research Foundation of State University of New York under the sponsorship of the U.S. Environmental Protection Agency. This report covers the period July 1, 1977 to August 31, 1978, and work was completed as of November 30, 1978.

CONTENTS

Foreword iii

Abstract iv

1. Introduction 1
2. Basic Ideas 4
3. Selection of Numerical Integration Programs 13
4. Runge-Kutta Methods 18
5. Adams Methods 23
6. Stiff Methods 35
7. Using the Codes 44
8. Examples 56

References 62

Appendices

A. Runge-Kutta Programs 64
B. Adams Method Programs 84
C. Stiff Method Programs 112
D. Solving Partial Differential Equations 145

SECTION 1
INTRODUCTION

The modeling of basin ecosystems is an immense undertaking which is most efficiently conducted by a team of professionals who bring together a variety of technical expertise. Mutual collaboration on such projects is aimed at producing a unified representation of an entire system. Too often, however, the greatest difficulties in such a "systems approach" are encountered not within any single discipline, but at the interfaces between disciplines.

One such interface, where a large gap exists, is between ecologists and mathematicians. The majority of ecologists have been unsuccessful in articulating their problems in a form understandable to mathematicians. Mathematicians, on the other hand, while equipped with the tools to solve many ecological problems, have little or no familiarity with the systems under study, and are equally unable to bridge the gap between problem articulation and solution.

This report and the accompanying software are an attempt to ease this problem for the ecologist by presenting methods for the numerical solution of ordinary differential equations, methods which can be used with ease and confidence without need for a sophisticated understanding of the techniques of numerical analysis.

Other bridges need to be built. This work does not address the problem of translating intimate knowledge of an ecosystem into a system of differential equations. Fortunately, numerous examples of the necessary techniques have appeared in the ecological literature as the importance of these endeavors has become increasingly recognized. Patten(1971,1972,1975, 1976) has published several volumes which provide instruction in the methods of ecological model construction, and which present examples from a variety of ecosystems.

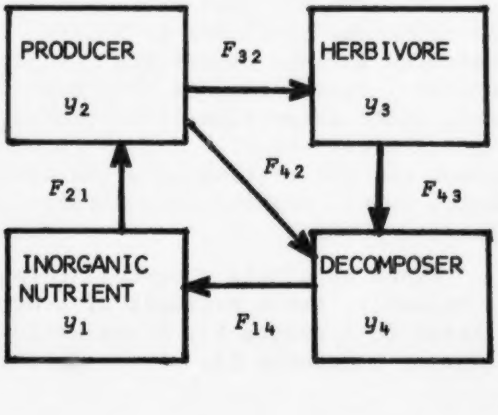


Figure 1-1. Simple Ecosystem Model. The y_i represent components of the system and the F_{ij} represent the material fluxes between components.

For illustrative purposes, a simple ecosystem model is presented. Suppose that this ecosystem is adequately described by the existence of an inorganic nutrient, a producer, an herbivore, and a decomposer. The relationships among the various components are depicted in Figure 1-1.

Given the box and arrow diagram of Figure 1-1, the differential equations describing the time dependence of the components may be written as

$$\begin{aligned} y_1' &= F_{14} - F_{21} & \qquad y_3' &= F_{32} - F_{43} \\ y_2' &= F_{21} - F_{32} - F_{42} & \qquad y_4' &= F_{42} + F_{43} - F_{14} \end{aligned} \tag{1-1}$$

where the primes denote differentiation with respect to time.

The model is not yet complete, for the mechanism which controls the material flows has not been specified. Suppose that each material flux is regulated only by the donating component and, furthermore, is proportional to that component. This assumption specifies the model described by

$$\begin{aligned} y_1' &= a_{14}y_4 - a_{21}y_1 \\ y_2' &= a_{21}y_1 - (a_{32}+a_{42})y_2 \\ y_3' &= a_{32}y_2 - a_{43}y_3 \\ y_4' &= a_{42}y_2 + a_{43}y_3 - a_{14}y_4 \end{aligned} \tag{1-2}$$

where the a_{ij} are positive constants

If this model is to describe a real system, the constants must be determined. Typically, the ecologist will have observations of the values of the components over some range of times, and will have some idea of the approximate values of the constant parameters. His first task, then, is to refine the values of these parameters in order to provide the best fit between the model and the empirical data. The adequacy of the model will be judged by the degree of agreement which can be obtained.

To carry out this fitting procedure, the ecologist must be able to solve the equations (1-2) for a variety of choices of the parameters. The codes presented in this report are intended to provide the tool needed for this purpose. These routines have been designed to be easily implemented and used, while automatically providing a sophisticated and efficient handling of the numerical solution. They have been written in ANSI FORTRAN IV and, except for the setting of a few machine dependent constants in DATA statements, should require no modification to run on any system which provides standard FORTRAN.

Since no single numerical integration method can solve all problems efficiently, three packages of programs are presented: Runge-Kutta Programs (listed in Appendix A), Adams Method Programs (Appendix B), and Stiff Method Programs (Appendix C). Guidelines for choosing the method best suited to a particular problem are stated in Section 7, which constitutes a user's manual for the codes. Furthermore, the three methods have been designed to be easily interchangeable, so that a little experimentation to discover the best choice should not be inconvenient.

For most users, it will suffice to read Section 2, which contains some basic definitions, and Section 7, which explains the use of the codes. Fairly complete instructions for use are also contained as comments within the program listings in Appendices A, B, and C. Section 8 presents some simple examples illustrating the performance of the codes.

Section 3 is preparatory for Sections 4, 5, and 6, which provide detailed descriptions of, respectively, the Runge-Kutta Programs, the Adams Method Programs, and the Stiff Method Programs. These sections should be consulted when a more thorough understanding of the programs is necessary, such as when modifications to the codes are being considered.

Appendix D contains a brief and elementary account of the manner in which the programs provided here can be utilized for the solution of some partial differential equation systems appearing in basin ecosystem models.

SECTION 2

BASIC IDEAS

This section presents a number of definitions and concepts pertaining to the numerical solution of ordinary differential equations. The aim is to provide only that information which is truly necessary for understanding the codes and for using them properly. Accordingly, a number of technical points are ignored or glossed over. More rigorous and complete treatments of these matters may be found in a number of textbooks, such as Gear(1971) and Shampine(1975).

§2.1 *The Initial Value Problem*

The codes presented in this report are designed for the numerical solution of the initial value problem for a system of first order ordinary differential equations,

$$\begin{aligned} y_1' &= f_1(t,y_1,y_2,\ldots,y_n) \\ y_2' &= f_2(t,y_1,y_2,\ldots,y_n) \\ &\;\;\vdots \\ y_n' &= f_n(t,y_1,y_2,\ldots,y_n) \end{aligned} \tag{2-1}$$

$$y_1(t_0) = y_{10},\quad y_2(t_0) = y_{20},\quad \ldots,\quad y_n(t_0) = y_{n0}$$

where the prime denotes differentiation with respect to the independent variable t. The independent variable will often be referred to as "time", which is its actual interpretation in many applications.

Such a problem is compactly described using a vector notation

$$\vec{y} = (y_1,y_2,\ldots,y_n)$$

$$\vec{f}(t,\vec{y}) = (f_1(t,\vec{y}),f_2(t,\vec{y}),\ldots,f_n(t,\vec{y}))$$

in terms of which (2-1) may be written as

$$\vec{y}'(t) = \vec{f}(t,\vec{y}) \qquad \vec{y}(t_0) = \vec{y}_0 \tag{2-2}$$

In discussing numerical integration procedures, it occasionally will be necessary to refer to the *Jacobian matrix* of $\vec{f}$. This is the n × n matrix

$$J\vec{f}(t,\vec{y}) = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_n} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \cdots & \frac{\partial f_2}{\partial y_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & \cdots & \frac{\partial f_n}{\partial y_n} \end{bmatrix} \qquad (2\text{-}3)$$

where all the partial derivatives are evaluated at $(t,\vec{y})$.

The numerical methods presented here apply only to the solution of first order equations. To solve the initial value problem for a higher order differential equation,

$$\begin{gathered} y^{(n)}(t) = f\left(t,y,y',y'',\ldots,y^{(n-1)}\right) \\ y(t_0) = a_0,\ \ y'(t_0) = a_1,\ \ y''(t_0) = a_2,\ \ \ldots,\ \ y^{(n-1)}(t_0) = a_{n-1} \end{gathered} \qquad (2\text{-}4)$$

it is necessary to restate the problem as a system of first order equations. One way of doing this is to define variables

$$u_k(t) = y^{(k)}(t),\ \ k = 1,\ 2,\ \ldots,\ n-1$$

in terms of which (2-4) is expressed as

$$\begin{aligned} y' &= u_1 \\ u_1' &= u_2 \\ u_2' &= u_3 \\ &\vdots \\ u_{n-2}' &= u_{n-1} \\ u_{n-1}' &= f(t,y,u_1,u_2,\ldots,u_{n-1}) \end{aligned}$$

$$y(t_0) = a_0,\ \ u_1(t_0) = a_1,\ \ u_2(t_0) = a_2,\ \ \ldots,\ \ u_{n-1}(t_0) = a_{n-1}$$

which is a problem of the form (2-1).

It should be understood that this procedure for reducing (2-4) to a system of first order equations is not unique. For a particular problem, physical or mathematical considerations may suggest a different choice of auxiliary variables which is more meaningful and appropriate.

§2.2 *Numerical Solution of the Initial Value Problem*

For purposes of analysis and explication, it is generally sufficient to discuss the single-equation problem

$$y' = f(t,y) \qquad y(t_0) = y_0 \tag{2-5}$$

Because of the formal resemblance between (2-5) and (2-2), insights gained from discussion of (2-5) can usually be taken over in obvious fashion to the n-equation problem (2-1).

A numerical solution of (2-5) consists of finding approximations y_0, y_1, y_2, ... to the solution $y(t)$ on a sequence of discrete *mesh points* t_0, t_1, t_2, ... in an interval $[t_0,T]$. The spacing between adjacent mesh points is called the stepsize and is denoted by h. The mesh points are usually not equally spaced; when the stepsize referred to is not clear from the context, it is distinguished by use of a subscript:

$$h_i = t_i - t_{i-1}$$

Starting with the given initial point (t_0,y_0), the numerical solution proceeds step by step, with the solution at each mesh point being calculated by some *numerical integration method*, which specifies the solution in terms of $f(t,y)$ and solution values already found at previous mesh points. Integration methods may be classified by the number of previous values which they require: an ℓ-step method utilizes the solution values at the ℓ preceding mesh points. A *single-step method* (ℓ=1) is also called *self-starting*, since it can make the first step of the solution using only the information given in the problem statement (2-5). A *multistep method* (ℓ>1) cannot be used to begin the solution, but it may be used to continue after a sufficient number of points have been calculated by a different method. The simplest single-step methods are

$$\text{Euler method:} \quad y_i = y_{i-1} + h_i f(t_{i-1},y_{i-1}) \tag{2-6a}$$

$$\text{backward Euler method:} \quad y_i = y_{i-1} + h_i f(t_i,y_i) \tag{2-6b}$$

Integration methods may be further classified as explicit or implicit. An *explicit method*, such as (2-6a), expresses the solution directly in terms of previously calculated values, while an *implicit method*, such as (2-6b), specifies the solution implicitly by stating an algebraic equation which it must satisfy.

The problem (2-5) is *well-posed* over the interval $[t_0,T]$ if small changes in the equation or in the initial condition produce only small changes in the solution. This is a more stringent condition than the mere existence of a unique solution.

Since any numerical computation must inevitably introduce small errors, well-posedness is necessary for (2-5) to be amenable to numerical solution. Otherwise, these small errors would completely change the character of the computed solution. Although it is possible to artificially construct problems which are not well-posed, such problems are unlikely to arise as models of real systems, since such models must display continuous behavior within the uncertainites of the empirical data upon which they are constructed. In any case, well-posedness need not be a serious *a priori* concern of the modeler, since its lack will become apparent from the anomalous behavior of the solution as the parameters of the model are varied.

§2.3 *Errors in Numerical Solution*

The *error* (or *global error*) in a point of the numerical solution is the amount by which it differs from the correct value,

$$\varepsilon_i = y(t_i) - y_i \tag{2-7}$$

where $y(t)$ is the exact solution of (2-5).

Errors in the numerical solution arise from two sources: truncation errors and roundoff errors. Roundoff errors are those which occur because arithmetic is not performed exactly. Even if the arithmetic were exact, truncation errors would occur because the integration method itself is only an approximation. All the methods discussed here are *convergent*, meaning that (assuming exact arithmetic) the error can be kept arbitrarily small by choosing the stepsizes sufficiently small.

The *local truncation error* for a step of the integration is the error which would be introduced at that step if the solution to the beginning of the step were exact:

$$\begin{aligned} &\tau_i = y(t_i) - y_i \\ &\text{given: } y_{i-1} = y(t_{i-1}),\ \ y_{i-2} = y(t_{i-2}),\ \ \ldots \end{aligned} \tag{2-8}$$

The form of the local truncation error is characteristic of the integration method being employed. A method is said to be of order k if the local truncation error is of order h^{k+1}. (This definition of the order of the method assumes that $y(t)$ possesses at least k continuous derivatives; if this is not the case for a particular problem, then the effective order of the method is reduces accordingly when applied to that problem.)

If the numerical solution is to be a useful approximation of $y(t)$, the global error must be kept within reasonable bounds throughout the interval of integration. In the codes, the error is controlled indirectly by choosing each stepsize small enough to keep a quantity called the *local error* smaller than limits supplied by the user. To understand the meaning of local error, consider all the solution curves in the (t,y) plane, of the differential equation in (2-5). For a well-posed problem, these curves do not intersect in any region of interest. The initial condition in (2-5) selects as the solution the unique curve $y(t)$ which contains the point (t_0,y_0). Suppose that the numerical solution has proceeded to t_{i-1}. Because y_{i-1} contains some error, it does not lie on $y(t)$, but rather on the nearby solution curve $u(t)$ specified by $u(t_{i-1})=y_{i-1}$. In making the step to t_i, the numerical method will then be following $u(t)$ instead of $y(t)$. The *local error* in the step from t_{i-1} to t_i is defined to be

$$\eta_i = u(t_i) - y_i \tag{2-9}$$

and is a measure of the amount by which the step fails to follow the solution curve on which it starts: η_i may be thought of as the truncation error of the method as applied to $u(t)$ instead of to $y(t)$.

Formally, the local errors may be made arbitrarily small by using sufficiently small stepsizes. However, this cannot be done in practice, since too small a stepsize will produce roundoff errors which exceed the local error. Thus, the accuracy achievable in the computed solution is ultimately limited by the precision of the arithmetic.

Although limiting the local error as just described does place a bound on the global error, the size of this bound depends on the characteristics of the problem being solved. If $\partial f/\partial y>0$, then the curves $u(t)$ and $y(t)$ diverge with increasing t, so that the global errors ε_i, ε_{i+1}, ... will increase even if the local errors η_i, η_{i+1}, ... vanish. A problem with this behavior is termed *unstable*. If the nearby solution curves diverge rapidly from the desired solution, the global error at the end of the integration interval may be expected to be substantially larger than the limit placed on the local error. Confining the global error within a given bound will then require that the local errors be limited to much smaller values, in order to keep the computed solution away from those curves which move too far away from $y(t)$ during the integration interval. The need to solve a severely unstable problem, $\partial f/\partial y>>0$, is unlikely to arise in practice, since any uncertainty in empirical initial conditions necessarily encompasses widely differing solutions: in a practical sense, such a problem is not well-posed.

If, on the other hand, $\partial f/\partial y<0$, then the nearby solution curves converge toward $y(t)$. Under these circumstances, the effect of previous errors is damped out as integration proceeds, and the global errors may be expected to be comparable in magnitude to the limit placed on local errors. Such problems are termed *stable*.

For a given method, the expense of integrating (2-5) is proportional to the number of steps required to span the integration interval $[t_0,T]$. For most problems, the solution curves neither diverge rapidly nor converge rapidly. For these problems, $\eta_i \approx \tau_i$ and the stepsize necessary for an accurate integration may be plausibly estimated from the form of $y(t)$, i.e., on the need for keeping the truncation errors small. As discussed above, a severely unstable problem requires a much smaller stepsize, and so is much more expensive to integrate accurately. It might be supposed that the opposite case of a highly stable problem, $\partial f/\partial y \ll 0$, would present no particular difficulty. Unfortunately, this is not true, for reasons which are discussed next.

§2.4 *Stability of an Integration Method*

If $\partial f/\partial y<0$, the problem (2-5) is stable, so that errors in the numerical solution are normally expected to be damped out as the integration proceeds. The point to be made here is that this is untrue unless the stepsize is kept smaller than a maximum value which depends both upon the integration method and upon the problem being solved.

For purposes of illustration, advance the solution of (2-5) from t_i to t_{i+1} by the Euler method (2-6a),

$$y_{i+1} = y_i + hf(t_i,y_i) \tag{2-10}$$

To determine the fate of an error ε_i in y_i, substitute

$$\begin{aligned} y_{i+1} &= y(t_{i+1}) - \varepsilon_{i+1} \\ y_i &= y(t_i) - \varepsilon_i \end{aligned} \tag{2-11}$$

into (2-10), obtaining

$$\varepsilon_{i+1} = \varepsilon_i + y(t_{i+1}) - y(t_i) - hf(t_i,y(t_i)-\varepsilon_i)$$

or, to a first order approximation in ε_i,

$$\varepsilon_{i+1} = \varepsilon_i(1+h\frac{\partial f}{\partial y}) + \tau_{i+1} \tag{2-12}$$

where the local truncation error is as defined by (2-8). Thus, if the error ε_i is not to be magnified in making the step, it is necessary that h be chosen small enough that

$$\left| 1 + h\frac{\partial f}{\partial y} \right| < 1$$

Remembering that $\partial f/\partial y<0$, this may be rewritten as the condition that

$$h < \frac{2}{|\partial f/\partial y|}$$

This limitation depends on the integration method used. Using a different integration method would impose a different maximum stepsize, but nearly all methods have some limitation of this sort. When this maximum stepsize is exceeded, the method is said to be *unstable*: errors occurring in one step tend to grow in subsequent steps, even though the problem itself is stable.

For the solution of a system of equations (2-2), the rôle of $\partial f/\partial y$ is played by the eigenvalues of the Jacobian matrix Jf. Since these eigenvalues are, in general, complex, it is customary to describe the stability of an integration method by reference to the standard problem

$$\begin{aligned} y'(t) &= \lambda y \\ y(t_0) &= y_0 \end{aligned} \qquad (2\text{-}13)$$

where y is complex and λ is a complex constant. When $\text{Re}\lambda<0$, this problem is stable. For the Euler method (2-10), the analysis proceeds precisely as before, with λ in place of $\partial f/\partial y$, except that the approximation in (2-12) is now exact. The stability condition for the Euler method is thus

$$\left| 1 + h\lambda \right| < 1 \qquad (2\text{-}14)$$

The *region of stability* of an integration method is that portion of the left half of the complex plane in which $h\lambda$ must lie if the method is to be stable when applied to (2-13). Each integration method has its own characteristic stability region. The stability region of the Euler method, as given by (2-14), is the interior of the unit circle centered at $-1+0i$.

The stepsize is also bounded by the need to keep the local truncation error smaller than an acceptable limit. If, as is normally the case, this bound is more stringent than that imposed by the stability condition, *the stepsize is limited for reasons of accuracy*. If, on the other hand, the stability condition imposes the more stringent bound, *the stepsize is limited for reasons of stability*.

Stiff problems are those for which most integration methods limit h for reasons of stability. Typically, the solution is smooth and slowly varying, and so would appear easy to follow; but $\partial f/\partial y \ll 0$, requiring a much smaller stepsize than is needed to keep the local truncation error within bounds. Thus, a curve which should be easy to compute requires much more work than expected. In extreme cases, the stepsize limitation may be so severe that roundoff errors make it impossible to proceed at all.

The difficulty presented by stiff problems may be pictured as follows: The computed points do not lie exactly on $y(t)$, but rather lie on nearby solution curves which, because of their rapid convergence toward $y(t)$, vary much more rapidly than $y(t)$ itself. Consequently, for most integration methods,

$$|\eta_i| \gg |\tau_i| \qquad (2\text{-}15)$$

and the stepsize must be small encough to follow these rapid variations, and cannot have the much larger size which would be sufficient to follow the slowly varying $y(t)$. Although controlling the local error closely enough will prevent the method from becoming unstable, the integration will be very expensive unless it is possible to employ a method for which the inequality (2-15) is not so severe.

It is rare that an initial value problem is stiff over the entire interval of integration. Usually, the solution has an initial transient which decays rapidly to approach a slowly varying curve. Until this transient has decayed to an insignificant value, the stepsize limitation is one of accuracy rather than one of stability, and the problem is not stiff over this initial interval.

In order to solve stiff problems efficiently, it is necessary to seek methods which do not limit h for reasons of stability. An example of such a method is the backward Euler method (2-6b),

$$y_{i+1} = y_i + hf(t_{i+1},y_{i+1}) \tag{2-16}$$

For the standard problem (2-13), this implicit method yields a linear equation which is easily solved to give

$$y_{i+1} = \frac{y_i}{1 - h\lambda}$$

The substitution (2-11) then gives

$$\varepsilon_{i+1} = \frac{\varepsilon_i}{1 - h\lambda} + \tau_{i+1}$$

Thus, the stability condition for the backward Euler method is

$$|\ 1 - h\lambda\ | > 1$$

and the stability region occupies the entire left half of the complex plane.

Such a method is suitable for solving stiff problems, since it imposes no stability limitation on the stepsize, which is then limited only for reasons of accuracy. No explicit method has such favorable stability properties, but applying an implicit method entails the additional labor of solving a (generally nonlinear) algebraic equation at each step. For a sufficiently stiff problem, the ability to use a larger stepsize will more than compensate for this additional work.

One approach to solving (2-16) is to refine an initial approximation $y_{i+1(0)}$ by successive iterations

$$y_{i+1(m+1)} = y_i + hf(t_{i+1},y_{i+1(m)})$$

Unfortunately, this simple iteration will not converge for stiff problems. Define

$$\delta_m = y_{i+1} - y_{i+1(m)}$$

the amount by which the mth iterate differs from the solution. Then it is easily found that, to a first order approximation in δ_m,

$$\delta_{m+1} = \delta_m h\frac{\partial f}{\partial y}$$

Therefore, convergence is not obtained unless $|h(\partial f/\partial y)| < 1$. For stiff problems, $\partial f/\partial y \ll 0$, this implies a limitation on h of the sort which must be avoided.

To solve (2-16) without this limitation, it is necessary to employ a Newton iteration,

$$y_{i+1(m+1)} = y_{i+1(m)} - \left(1-h\frac{\partial f}{\partial y}\right)^{-1}\cdot\left(y_{i+1(m)}-hf(t_{i+1},y_{i+1(m)})-y_i\right) \tag{2-17}$$

For nearly all practical problems $\partial f/\partial y$ varies only slowly, and the Newton iteration will converge for fairly large values of h.

If an n-equation system (2-2) is being solved, then (2-17) is a vector equation in which $(1-h(\partial f/\partial y))^{-1}$ is replaced by the $n \times n$ matrix

$$(I - hJ\vec{f})^{-1}$$

where I is the identity matrix and $J\vec{f}$ is the Jacobian matrix of $\vec{f}$. The need to evaluate the n^2 components of the Jacobian and to invert an $n \times n$ matrix, which is common to all methods suitable for stiff equations, makes the numerical solution of very stiff problems generally intractable for large n. Nonetheless, these methods can be very useful for systems of moderate size.

SECTION 3

SELECTION OF NUMERICAL INTEGRATION PROGRAMS

The foremost consideration in the construction of a numerical integration routine is that of *efficiency*: of two programs, the more efficient requires less computing to solve the same problem. The efficiency of a program depends not only on the basic integration method used, but also on the particular coding which is used to implement that method. Thus, comparisons of efficiency can generally be made only between the complete codes, and not between the integration methods themselves.

Efficiency, however, cannot be the only criterion in constructing a practical code. A user will certainly desire that the code be *reliable*: it should not fail on routine problems, and it should not return erroneous results while appearing to operate normally. Furthermore, the code should be *easy to use*, requiring little more than a statement of the problem to be solved. While the user must be concerned with the nature of his problem, a good code should automatically handle all concerns which arise in the solution process. In selecting the codes to be incorporated in this report, attention was restricted to those existing codes which meet these criteria of reliability and ease of use. While no code is equally efficient for all problems, each of the three codes presented here may be regarded as representing state-of-the-art performance when applied to those problems for which it is intended.

§3.1 *General Considerations and Terminology*

The cost of solving (2-1) numerically may be viewed as

$$\text{cost} = (\text{number of steps required}) \times (\text{amount of calculation/step})$$

For a given integration method, the amount of calculation per step is fixed, and the cost is minimized by choosing each stepsize no smaller than is necessary to achieve the desired accuracy in the solution. If the program is to be reliable and easy to use, it must contain provision for choosing this optimum stepsize automatically. As mentioned in §2.3, the programs given here select the stepsize to keep the local error at each step smaller than some limit requested by the user. As the integration proceeds, this stepsize is increased or decreased to reflect the local requirements of accuracy. These programs are thus called *variable-stepsize* codes.

Of two methods, that having the smaller truncation error will, *ceteris paribus*, be able to solve the problem at lower cost by using larger steps. However, the construction of an efficient variable-stepsize code requires the ability to form a reliable estimate of the local error without greatly increasing the amount of calculation per step. Consequently, a number of methods which are excellent in the sense of having small truncation errors, are unsuitable because they do not allow a cheap estimation of the local error.

If the solution is sufficiently smooth, the use of a higher order integration method will, because of the smaller truncation error, allow use of a larger stepsize. Although higher order methods require more computation for each step, the stepsize can usually be sufficiently larger to produce an overall increase in efficiency. However, the use of a high order method for a problem with discontinuities in a low order derivative will be less efficient, since no increase of stepsize will then be possible. For this reason, high order methods are best incorporated into programs which can automatically choose an order which is suitable for the particular problem being solved. Programs which can do this are termed *variable order* codes.

Part of the calculation performed in each step of the integration is in the form of *function evaluations*, the evaluation of the derivatives in (2-1) at a given $(t,\vec{y})$. The remaining calculation performed in the step is termed *overhead*. The division of labor between function evaluations and overhead varies widely among integration methods. Some methods require many function evaluations, but relatively small overhead; while others employ fewer function evaluations at a cost of increased overhead. Methods of the former type will be more efficient for problems whose function evaluations are cheap, while those of the latter type are preferable for problems whose function evaluations are expensive.

Increasing the stringency of the local error tolerances will always increase the expense of the integration, since smaller stepsizes must be used. However, the increase in cost will occur at different rates for different methods, so that the method of choice for use at moderate accuracy may be different from that which is best for extreme accuracy.

To monitor the accuracy of the solution and to choose an appropriate stepsize, the code must estimate the local error at each step. In a typical case, the local error may be estimated as follows: using the same stepsize h,

compute y_i by a method of order k (less accurate solution)
compute $\hat{y}_i$ by a method of order $k+1$ (more accurate solution)
estimate local error in y_i as $\eta_i = \hat{y}_i - y_i$

Estimated in this way, $\eta_i = O(h^{k+1})$ and so is the local error in the less accurate solution. The stepsize control is thus based on the less accurate solution. However, since the more accurate solution $\hat{y}_i$ has already been found, it is reasonable to accept it as the computed value and to use this value when continuing the integration. In effect, one improves the accuracy of y_i by adding the estimated local error to it. This procedure, which is called *local extrapolation*, may be used whenever there is good reason to believe that the local error estimate is quite accurate. It may be noted that the above method for estimating η_i is practical only when the calculation of $\hat{y}_i$ can be done using very little computation beyond that already needed for y_i: integration methods which do not allow this cannot economically estimate the local error in this way.

§3.2 *Control of Stepsize*

As discussed above, achieving maximum efficiency for a given integration method requires that each stepsize be chosen as large as is permitted by the constraint

$$\eta < E$$

where η is the absolute value of the local error in the step, and E is an error tolerance specified by the user of the code.

Suppose that a method of order k is being used, so that $\eta = O(h^{k+1})$. Then the basic procedure for error control and stepsize adjustment is as follows:

1.) Perform the step with the (previously chosen) stepsize h and estimate η.

2.) If $\eta < E$, then the stepsize was smaller than required by the error control. If $\eta > E$, the stepsize was too large. Since $\eta = O(h^{k+1})$, the stepsize h' which will exactly meet the error criterion may be estimated as

$$h' = \left[\frac{E}{\eta}\right]^{\frac{1}{k+1}} \cdot h \qquad (3\text{-}1)$$

3.) If $\eta > E$, then the step did not meet the required error tolerance, so repeat it with the new (smaller) stepsize h'. If $\eta \leq E$, the step was successful, so use h' in attempting the next step.

In practice, h' must be chosen smaller than (3-1) indicates. Because of uncertainties in η and changes in the requirements of the solution from step to step, strict adherence to (3-1) would produce frequent step failures, $\eta > E$. Since each failed step must be repeated, efficiency will be lost unless the number of such failures is kept small by a conservative choice of stepsize. This problem is handled somewhat differently in the various codes, so the details are reserved for the detailed code descriptions in Sections 4, 5, and 6. The choice of an initial stepsize for beginning the integration will also be described there.

§3.3 *Choice of the Codes*

The codes chosen for inclusion in this report are incorporated into three packages of subroutines:

Runge-Kutta programs: A fixed-order, variable-stepsize code employing the single-step Runge-Kutta-Fehlberg fourth-fifth order method. Local extrapolation is used, the computed solution coming from the fifth order formula. This program is most efficient for nonstiff problems whose function evaluations are cheap, and for which extreme accuracy is not requested.

Adams method programs: A variable-order, variable-stepsize code employing the multistep Adams PECE methods of orders 1 through 12.

Local extrapolation is used, so that the effective orders are 2 through 13. This program is most efficient for nonstiff problems whose function evaluations are fairly expensive or for which high accuracy is requested.

Stiff method programs: A variable-order, variable-stepsize code employing the multistep backward differentiation methods of orders 1 through 6. This program is intended only for solving stiff problems. When applied to nonstiff problems, it will be less efficient, and often less accurate, than the other two methods.

The integration routine for the Runge-Kutta programs, RKFS, is a modified version of that given in Shampine(1976). On the basis of tests reported there, and after further tests comparing RKFS with some Runge-Kutta methods recently proposed by Bettis(1978), RKFS was selected as the most efficient of those presently existing programs which utilize Runge-Kutta methods of fifth order or less. The Runge-Kutta-Fehlberg method is described in Section 4, which also describes the coding of the Runge-Kutta programs.

The power and efficiency of variable-order, variable-stepsize codes employing Adams methods was first recognized in codes written by Krogh(1969) and Gear(1971,1971a). The code STEP given here is taken from Shampine(1975), and represents the most sophisticated level of development of such codes. Section 5 explains the Adams PECE methods, and also describes the coding of the Adams method programs.

As discussed in §2.4, stiff problems require special methods for their efficient solution. The backward differentiation methods which are used for the stiff method programs are described in Section 6. The program GEAR which incorporates these methods is a modified version of an algorithm originally developed by Gear(1971,1971a), the chief modification being a more conservative stepsize selection procedure. While this is presented as a general-purpose code for stiff problems, it cannot, for reasons mentioned in §2.4, be expected to solve very large systems of equations. The handling of very large stiff problems generally requires much special-purpose coding tailored to the particular problem, and so falls outside the purview of this report.

One class of methods not represented here is that of the *extrapolation methods*, which are single-step methods suitable for incorporation into a variable-order, variable-stepsize code. The highest state of their development in a practical code is represented by the program DODES described in Schryer(1975). Such codes are best suited for problems whose function evaluations are cheap, but for which high accuracy is required. For these problems, extrapolation codes can exceed the efficiency of Adams codes, which are, in turn, more efficient than Runge-Kutta codes for work at high accuracies. However, the overall reliability of extrapolation codes is somewhat uncertain because of unresolved theoretical and practical questions about their behavior upon encountering stiffness or lack of smoothness in the solution. The chief reason that they have been excluded from further consideration here is that the efficiency of all extrapolation methods tends to be very sensitive to the choice of output points: since the average user desires to specify the particular points at which solution values are to be returned, he is likely to find this feature a serious inconvenience.

§3.4 *Limitations of Precision*

As was seen in §2.3, the accuracy which can be achieved is limited by the precision of the arithmetic used to perform the computations. The precision of the arithmetic is most conveniently described by the computer unit roundoff error u, which is the smallest positive value for which $1.+u > 1.$ in floating point arithmetic. Typical values of u for single precision arithmetic range from 9.54×10^{-7} for the IBM 360 to 7.11×10^{-15} for the CDC 6600.

During the numerical integration, it is clear that one cannot make progress with a stepsize so small that $t+h=t$ in floating point arithmetic. Thus, each stepsize must be chosen to be at least as large as tu, where t is the current value of the independent variable. In fact, h must be chosen somewhat larger than this in order to allow the stepsize adjustment mechanism some range in which to operate. All of the codes in this report require h to be at least as large as $4ut$. In the case of the Runge-Kutta code, the properties of the integration formula require a much larger limit, and h is limited to a minimum value of $26ut$.

Similarly, it is not possible to require the solution to have an error smaller than yu, which is the error inherent in its correctly rounded value. The error tolerance E must be somewhat larger than this ultimate limit if stepsize adjustment is to work properly. The codes all require that E be at least as large as $4uy$, where y is the current value of the solution.

If, at any point during the integration, the codes find that the user's requested error tolerance is too small, or that meeting that tolerance requires too small a stepsize, the integration is stopped and the user is informed of what (larger) tolerance will allow a successful continuation. This provision is necessary for the reliability of the codes, so that they will not return meaningless or inaccurate values heavily contaminated by roundoff errors.

SECTION 4

RUNGE-KUTTA METHODS

An r-stage explicit Runge-Kutta method advances the solution of (2-5) according to

$$y_{i+1} = y_i + \sum_{\ell=1}^{r} \gamma_\ell k_\ell \tag{4-1}$$

where

$$k_1 = hf(t_i, y_i)$$

$$k_\ell = hf\Big(t_i + \alpha_\ell h \,,\, y_i + \sum_{j=1}^{\ell-1} \beta_{\ell j} k_j\Big) \qquad \ell = 2, 3, \ldots, r$$

$$\alpha_\ell = \sum_{j=1}^{\ell-1} \beta_{\ell j} \qquad\qquad 1 = \sum_{\ell=1}^{r} \gamma_\ell$$

The number of stages, r, is the number of evaluations of f needed to perform one step. If the β and γ coefficients are chosen so that τ_{i+1} is of order h^{s+1}, then the method is of order s.

A Runge-Kutta is conventionally specified by presenting the coefficients in a tableau:

$$\begin{array}{c|cccccc}
0 & & & & & & \\
\alpha_2 & \beta_{21} & & & & & \\
\alpha_3 & \beta_{31} & \beta_{32} & & & & \\
\vdots & \vdots & \vdots & & & & \\
\alpha_r & \beta_{r1} & \beta_{r2} & \beta_{r3} & \cdots & \beta_{r,r-1} & \\
\hline
 & \gamma_1 & \gamma_2 & \gamma_3 & \cdots & \gamma_{r-1} & \gamma_r
\end{array}$$

The program RKFS employs the following six-stage, fifth order method derived by Fehlberg(1969):

$$\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{4} & \frac{1}{4} & & & & & \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} & & & & \\
\frac{12}{13} & \frac{1932}{2197} & \frac{-7200}{2197} & \frac{7296}{2197} & & & \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & \frac{-845}{4104} & & \\
\frac{1}{2} & \frac{-8}{27} & 2 & \frac{-3544}{2565} & \frac{1859}{4104} & \frac{-11}{40} & \\
\hline
 & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{28561}{56430} & \frac{-9}{50} & \frac{2}{55}
\end{array} \tag{4-2}$$

The utility of this method derives from the fact that it has embedded within it the five-stage, fourth order method given by the first five rows of the above table together with the γ coefficients

$$\begin{array}{c|ccccc}
\hline
 & \frac{25}{216} & 0 & \frac{1408}{2565} & \frac{2197}{4104} & -\frac{1}{5}
\end{array} \tag{4-3}$$

Thus, at a cost of six function evaluations, both a fifth order solution and a fourth order solution are obtained. The difference of these two results provides an estimate of the local error in the fourth order solution, and this estimate is used for error control and stepsize selection. However, the fifth order solution is the value actually accepted; i.e., local extrapolation is used.

The remainder of this section describes the implementation of this method in the code RKFS, and should be read in conjunction with the program listing in Appendix A and the instructions for use in Section 7. RKFS is very similar to the program in Shampine(1976), which reference should be consulted if more detailed information is needed.

Applying (4-1) to a system of n equations (2-2) is accomplished simply by interpreting the y's, f's, and k's as n-dimensional vectors. At each step of the integration, there is then a vector of local error estimates $\eta(j)$ to be compared with a vector $E(j)$ of local error tolerances, $j = 1, 2, \ldots, n$. The $E(j)$ are computed from user-specified relative and absolute error tolerances, as described in Section 7. For a successful step, it is required that $\eta(j) \leq E(j)$ for all components of the solution: $j = 1, 2, \ldots, n$. For stepsize control, it is necessary to form a single number which characterizes the maximum error occurring in any of the components. This is done by calculating

$$\rho = \max_{j=1,\ldots,n} \left| \frac{\eta(j)}{E(j)} \right| \tag{4-4}$$

so that $\rho<1$ for a successful step and $\rho>1$ for a failed step. The stepsize to be used for the next step, or to repeat the last step if it failed, is then computed as

$$h_{\text{new}} = h_{\text{old}} \times 0.9 \times \rho^{1/5} \tag{4-5}$$

where the factor 0.9 causes the new stepsize to be chosen conservatively in order to avoid unnecessary step failures. Since the assumptions underlying (4-5) are not valid if the solution is not sufficiently smooth, possible difficulties are avoided through limiting its operation by three constraints:

1.) Stepsize increase is limited to a factor of 5.

2.) Stepsize decrease is limited to a factor of 0.1.

3.) The stepsize is not allowed to increase for the step following one which failed on the first attempt.

While (4-5) specifies how the stepsize is to be adjusted during the integration, a separate procedure is needed to select a stepsize for beginning the integration. The starting stepsize need not be highly accurate, since it will be adjusted to an appropriate value by (4-5) after the first step is attempted. If the first step were taken with a constant approximation, $\vec{y}_1=\vec{y}_0$, then the error would be estimated as $h\vec{f}(t_0,\vec{y}_0)$. Estimating the error in a fourth order method as h^4 times the error in this "zeroth order" method, an appropriate starting stepsize may be estimated as

$$h = \min_{\substack{j=1,\ldots,n \\ E_0(j)\neq 0}} \left[\frac{E_0(j)}{|f_j(t_0,\vec{y}_0)|} \right]^{1/5} \tag{4-6}$$

where $E_0(j)=E_r|y_0(j)|+E_a$, E_r and E_a being the user-specified relative and absolute error tolerances RELERR and ABSERR, and $y_0(j)$ the initial value of the j-th component of the solution. (4-6) is supplemented by two rules:

1.) h is not allowed to be smaller than $26u\cdot\max(|t_0|,\Delta t)$, where Δt is the distance from t_0 to the first output point. This is the value used if all the $E_0(j)$ vanish.

2.) h cannot be larger than Δt. This deals with the case in which all the initial derivatives $f_j(t_0,\vec{y}_0)$ vanish.

RKFS performs a number of checks designed to protect the user from improper use of the code. Attempts at impossible precision are prevented by requiring that the relative error tolerance RELERR be at least as large as the machine dependent constant REMIN, whose value is set in a DATA statement.

A request for greater accuracy results in an error return signaled by IFLAG=3, and with RELERR set to the value of REMIN. The value of REMIN should not be set smaller than $4u$, where u is the computer unit roundoff error. It is further recommended that REMIN not be smaller than 10^{-12}, since it is more efficient to use Adams methods if greater accuracy than this is required.

To distinguish the values of the arguments $t+\alpha_4 h = t+(12/13)h$ and $t+\alpha_5 h = t+h$, which appear in the evaluations of k_4 and k_5 in (4-1), the stepsize is never allowed to be smaller than $26ut$. If the code finds it impossible to satisfy the error test using the smallest permissible stepsize, it returns with IFLAG=4 and with RELERR and ABSERR increased to values which should be achievable.

Unlike Adams codes, which can interpolate to obtain the solution at an arbitrary point, Runge-Kutta methods must adjust the stepsize so that the output point will be a mesh point of the integration. Unless some foresight is applied, the last step to the output point may be exceedingly small. This would impair the efficiency of the stepsize selection procedure (4-5), and might even require a step smaller than $26ut$. RKFS avoids this difficulty by looking ahead two steps of the current stepsize. If the output point lies within this interval, then the stepsize is adjusted as follows: If the output point is within $h/0.9$ if the current value of t, the stepsize is set to complete the integration in one step. Otherwise, the stepsize is set to half the distance to the output point, so that the integration will be completed in two steps.

While this procedure greatly reduces the adverse impact of output points on stepsize selection, inefficiency may still occur if the user requests a sequence of output points which are spaced much more closely than the natural stepsize which would be selected by (4-5). To inform the user of this, RKFS counts, in the variable KOP, the number of output intervals which are less than half the natural stepsize, and returns with IFLAG=7 if KOP exceeds 100.

Another possible difficulty, which should happen very seldom, occurs when the user requests an output point at distance less than $26ut$ from the previous output point. In this case, RKFS simply extrapolates the solution by the Euler method. No error check is made, and there is no special indication that this has been done.

In order to inform the user when a problem is unusually expensive to integrate, RKFS counts, in the variable NFE, the number of function evaluations which have been performed since the start of the integration. If NFE exceeds the value of MAXNFE, which is set in a DATA statement, RKFS returns with IFLAG=5 or IFLAG=6.

Since a common cause of an expensive integration is stiffness of the problem, RKFS incorporates a procedure to detect stiffness, as described in Shampine(1977). This test for stiffness is based on a first-second order pair of Runge-Kutta formulae which use the same function evaluations as (4-2) with the γ coefficients given by:

$$\begin{array}{|cccccc}
\text{first order method:} & & & & & \\
\hline
\frac{1094951}{13000000} & \frac{-2120820}{13000000} & \frac{9893169}{13000000} & \frac{5275998}{13000000} & \frac{-1715610}{13000000} & \frac{572312}{13000000}
\end{array} \qquad (4\text{-}7)$$

second order method:

$$\begin{array}{c|cccccc} & \frac{1815846}{13000000} & \frac{-2582209}{13000000} & \frac{9417746}{13000000} & \frac{5576389}{13000000} & \frac{-1839305}{13000000} & \frac{611533}{13000000} \end{array} \qquad (4\text{-}8)$$

This pair may be used to estimate the local error in the first order method, in precisely the manner described above for the fourth-fifth order pair (4-3) and (4-2). Although (4-7) and (4-8) are less accurate than (4-3) and (4-2), their stability regions are substantially larger than those of the fourth-fifth order methods. Thus, (4-7) and (4-8) are unlikely to meet the error test unless (4-2) and (4-3) have severely restricted the stepsize for reasons of stability. During the integration, RKFS examines successive blocks of 50 successful steps. In each block, the number of steps which pass the first-second order error test is counted in KST. If KST reaches 25, the problem is labeled as stiff, and it is unnecessary to test subsequent blocks. Since stiffness is not a problem unless it makes the integration too expensive, a finding of stiffness is not reported to the user unless there is an error return for too much work. Then, IFLAG=6 indicates that stiffness was found. While tests indicate that this method is quite effective in detecting severe stiffness, moderate stiffness may not be reported, particularly at stringent error tolerances.

SECTION 5
ADAMS METHODS

The solution of (2-5) at t_{n+1} may be written as

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t,y(t))\,dt \tag{5-1}$$

Define $P_{k\ell}(t)$ to be the (unique) k-1 degree polynomial which interpolates to the computed derivative values at the k mesh points $t_\ell, t_{\ell-1}, \ldots, t_{\ell+1-k}$:

$$P_{k\ell}(t_{\ell+1-j}) = f_{\ell+1-j} \qquad j = 1, 2, \ldots, k \tag{5-2}$$

where $f_i = f(t_i, y_i)$

The Adams-Bashforth method of order k approximates (5-1) by replacing $y(t_n)$ with y_n and $f(t,y(t))$ with $P_{kn}(t)$, so that y_{n+1} is given by

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} P_{kn}(t)\,dt \tag{5-3}$$

Since $P_{kn}(t)$ interpolates to known (i.e., previously computed) values, (5-3) is an explicit integration method. For k=1, it is the Euler method (2-6a). When the stepsize is constant, the truncation error in (5-3) is given by

$$\tau_{n+1} = \gamma_k h^{k+1} y^{(k+1)}(\xi) \tag{5-4}$$

where $\gamma_0 = 1$

$$\gamma_i = \frac{1}{i!}\int_0^1 s(s+1)(s+2)\cdots(s+i-1)\,ds \qquad i \geq 1$$

and ξ is in the interval $[t_n, t_{n+1}]$

The Adams-Moulton method of order k differs from the Adams-Bashforth method in replacing $f(t,y(t))$ with $P_{k,n+1}(t)$, so that y_{n+1} is given by

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} P_{k,n+1}(t)\, dt \tag{5-5}$$

When the stepsize is constant, the truncation error in (5-5) is

$$\tau_{n+1} = \gamma_k^* h^{k+1} y^{(k+1)}(\xi) \tag{5-6}$$

where $\gamma_0^* = 1$ and $\gamma_i^* = \gamma_i - \gamma_{i-1}$ for $i \geq 1$

Since $P_{k,n+1}(t)$ depends on y_{n+1}, (5-5) is an implicit method. For k=1, it is the backward Euler method (2-6b). Although applying the implicit Adams-Moulton method requires additional labor in solving (5-5) for y_{n+1}, the truncation error is smaller, and the stability region larger, than for the same order Adams-Bashforth method. Thus, the Adams-Moulton method is able to take larger steps while achieving the same accuracy, so that it is usually more efficient in spite of requiring more work for each step. Rather than attempting to solve (5-5) exactly, the code STEP uses the Adams-Bashforth and Adams-Moulton methods in combination as follows:

1. The Adams-Bashforth method of order k is used as a *predictor* to obtain a *predicted* solution

$$p_{n+1} = y_n + \int_{t_n}^{t_{n+1}} P_{kn}(t)\, dt \tag{5-7}$$

2. The derivative $f_{n+1}^p = f(t_{n+1}, p_{n+1})$ is evaluated using this predicted solution.

3. The Adams-Moulton method of order k+1 is used as a *corrector*, with the k-degree interpolating polynomial $P_{k+1,n+1}$ being approximated by $P_{k+1,n}^*$ given by

$$\begin{aligned} P_{k+1,n}^*(t_{n+1-j}) &= f_{n+1-j} \qquad j = 1, 2, \ldots, k \\ P_{k+1,n}^*(t_{n+1}) &= f_{n+1}^p \end{aligned} \tag{5-8}$$

so that the *corrected* solution is given by

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} P_{k+1,n}^*(t)\, dt \tag{5-9}$$

4. To complete the step, the derivative $f_{n+1} = f(t_{n+1}, y_{n+1})$ is evaluated using the corrected solution. This derivative is needed to begin the next step.

Because of the successive operations of *prediction*, derivative *evaluation*, *correction*, and final derivative *evaluation*, this method is conventionally referred to as a PECE method. Since it begins with the Adams--Bashforth method of order k, it will be called a PECE method of order k. However, the use of a corrector whose order is one higher than that of the predictor, which may be viewed as local extrapolation, causes this PECE method of order k to have an effective order k+1. Subroutine STEP employs the Adams PECE methods of orders 1 through 12, with local extrapolation giving effective orders of 2 through 13. During the integration, the stepsize and the order of the method are adjusted in order to proceed efficiently while keeping the local errors within the bounds specified by the user of the code.

The Adams methods may be expressed in many different equivalent forms, corresponding to different algebraic representations of the interpolating polynomials in (5-7) and (5-9). For practical computing, preference must be given to those forms which facilitate changes in stepsize and order, and which are economical of storage. The derivations of the formulae given below may be found in Shampine(1975). These derivations are quite lengthy and will not be repeated here.

The divided difference (or Newtonian) form of the polynomial $P_{kn}(t)$ which interpolates to $f_n, f_{n-1}, \dots, f_{n-k+1}$ is

$$\begin{aligned} P_{kn}(t) = {} & f[t_n] + (t-t_n)f[t_n,t_{n-1}] + (t-t_n)(t-t_{n-1})f[t_n,t_{n-1},t_{n-2}] \\ & + \cdots + (t-t_n)(t-t_{n-1})\cdots(t-t_{n-k+2})f[t_n,t_{n-1},\dots,t_{n-k+1}] \end{aligned} \tag{5-10}$$

where
$$f[t_j] = f_j$$

$$f[t_j,t_{j-1},\dots,t_{j-\ell}] = \sum_{i=0}^{\ell} f_{j-i} \prod_{\substack{s=0 \\ s\neq i}}^{\ell} \frac{1}{t_{j-i}-t_{j-s}}$$

The quantities $f[\cdots]$ are called *divided differences* because they obey the relation

$$f[t_j,t_{j-1},\dots,t_{j-\ell}] = \frac{f[t_j,t_{j-1},\dots,t_{j-\ell+1}] - f[t_{j-1},\dots,t_{j-\ell+1},t_{j-\ell}]}{t_j - t_{j-\ell}} \tag{5-11}$$

Starting with $f_n,f_{n-1},\dots,f_{n-k+1}$, (5-11) may be used to generate all the divided differences needed for the interpolating polynomial (5-10).

In the code STEP, the interpolating polynomials are represented in terms of modified divided differences defined by

$$\phi_1(n) = f[t_n] = f_n$$

$$\phi_i(n) = \psi_1(n)\psi_2(n)\cdots\psi_{i-1}(n)f[t_n, t_{n-1}, \ldots, t_{n-i+1}] \qquad i > 1 \tag{5-12}$$

where the ψ's are sums of stepsizes,

$$\psi_\ell(n) = h_n + h_{n-1} + \cdots + h_{n+1-\ell} \qquad \ell \geq 1$$

$$h_j = h_j - h_{j-1} \tag{5-13}$$

Defining the further notation

$$\beta_1(n+1) = 1$$

$$\beta_i(n+1) = \frac{\psi_1(n+1)\psi_2(n+1)\cdots\psi_{i-1}(n+1)}{\psi_1(n)\psi_2(n)\cdots\psi_{i-1}(n)} \qquad i > 1 \tag{5-14}$$

$$s = \frac{t - t_n}{h_{n+1}}$$

the interpolating polynomial (5-10) may be written as

$$P_{kn}(t) = \sum_{i=1}^{k} c_{in}(s)\beta_i(n+1)\phi_i(n)$$

where the coefficients are given by

$$c_{1n}(s) = 1$$

$$c_{2n}(s) = s$$

$$c_{in}(s) = s \prod_{\ell=1}^{i-2} \left[\frac{sh_{n+1} + \psi_\ell(n)}{\psi_{\ell+1}(n+1)}\right] \qquad i \geq 3$$

Then the predicted solution (5-7) may be written as

$$p_{n+1} = y_n + h_{n+1} \sum_{i=1}^{k} g_{i1}\phi_i^*(n) \tag{5-15}$$

where $\phi_i^*(n) = \beta_i(n+1)\phi_i(n)$ and the coefficients

$$g_{i1} = \int_0^1 c_{in}(s)\, ds$$

may be calculated recursively from

$$g_{1q} = \frac{1}{q}$$

$$g_{iq} = g_{i-1,q} - \alpha_{i-1}(n+1)g_{i-1,q+1} \qquad i \geq 2 \tag{5-16}$$

where
$$\alpha_j(n+1) = \frac{h_{n+1}}{\psi_j(n+1)} \qquad j \geq 1$$

The corrector polynomial $P^*_{k+1,n}$ interpolates to the same data as P_{kn} plus the value

$$f^p_{n+1} = f(t_{n+1}, p_{n+1})$$

and so is given by

$$P^*_{k+1,n}(t) = P_{kn}(t) + c_{k+1,n}(s)\phi^p_{k+1}(n+1)$$

where the superscript p indicates that the divided difference is generated using f^p_{n+1} rather than $f_{n+1} = f(t_{n+1}, y_{n+1})$. The corrected solution (5-9) is then

$$y_{n+1} = p_{n+1} + h_{n+1}g_{k+1,1}\phi^p_{k+1}(n+1) \tag{5-17}$$

where $\phi^p_{k+1}(n+1)$ may be calculated from

$$\phi^p_{k+1}(n+1) = f^p_{n+1} - \sum_{i=1}^{k} \phi_i^*(n)$$

To complete the step, it is necessary to compute the divided differences $\phi_i(n+1)$ to be used in beginning the next step. They are given by

$$\phi_1(n+1) = f_{n+1}$$

$$\phi_{i+1}(n+1) = \phi_i(n+1) - \phi_i^*(n)$$

A careful arrangement of the computation avoids the need for separate arrays to store the $\phi_i(n)$ and the $\phi_i^*(n)$. To achieve this economy of storage, STEP introduces intermediate quantities $\phi_i^e(n+1)$, and successively overwrites the divided differences,

$$\phi_i(n) \leftarrow \phi_i^*(n)$$

$$\phi_i^*(n) \leftarrow \phi_i^e(n+1)$$

$$\phi_i^e(n+1) \leftarrow \phi_i(n+1)$$

The calculations performed by STEP in advancing the solution from t_n to t_{n+1} are as follows:

computing $\beta_i(n+1)$ for $i=1,2,\ldots,k$ and g_{i1} for $i=1,2,\ldots,k+1$

predicting $\phi_i^*(n) = \beta_i(n+1)\phi_i(n) \quad i=1,2,\ldots,k$

$$p_{n+1} = y_n + h_{n+1} \sum_{i=1}^{k} g_{i1}\phi_i^*(n)$$

$$\phi_{k+1}^e(n+1) = 0$$

$$\phi_i^e(n+1) = \phi_{i+1}^e(n+1) + \phi_i^*(n) \qquad i=k,k-1,\ldots,1$$

evaluating $f_{n+1}^p = f(t_{n+1},p_{n+1})$

correcting $y_{n+1} = p_{n+1} + h_{n+1}g_{k+1,1}(f_{n+1}^p - \phi_1^e(n+1))$

evaluating $f_{n+1} = f(t_{n+1},y_{n+1})$

$$\phi_{k+1}(n+1) = f_{n+1} - \phi_1^e(n+1)$$

$$\phi_i(n+1) = \phi_i^e(n+1) + \phi_{k+1}(n+1) \qquad i=k,k-1,\ldots,1$$

This formulation is particularly convenient for changing the order of the method. Since the step taken with order k evaluates $\phi_{k+1}(n+1)$, this difference may be used to raise the order to k+1 for the next step. Lowering the order from k to k-1 is accomplished simply by not using $\phi_{k+1}(n+1)$ and $\phi_k(n+1)$ in starting the next step.

When the stepsize is constant, it is not necessary to recompute all the coefficients. Let n_s be the number of successive steps taken with the constant stepsize h, including the current step. Then

$$\psi_i(n+1) = \psi_i(n) = ih \qquad i = 1, 2, \ldots, n_s-1$$

$$\alpha_i(n+1) = \alpha_i(n) = \frac{1}{i} \qquad i = 1, 2, \ldots, n_s-1$$

$$\beta_i(n+1) = \beta_i(n) = 1 \qquad i = 1, 2, \ldots, n_s-1$$

and the remaining values are computed from

$$\psi_{n_s}(n+1) = n_s h$$

$$\psi_i(n+1) = \psi_{i-1}(n) + h \qquad i = n_s+1, \ldots, k$$

$$\alpha_{n_s}(n+1) = \frac{1}{n_s}$$

$$\alpha_i(n+1) = \frac{h}{\psi_i(n+1)} \qquad i = n_s+1, \ldots, k$$

$$\beta_{n_s}(n+1) = 1$$

$$\beta_i(n+1) = \beta_{i-1}(n+1) \frac{\psi_{i-1}(n+1)}{\psi_{i-1}(n)} \qquad i = n_s+1, \ldots, k$$

In STEP, the coefficients g_{i1} are stored in the array G(i), i=1,...,k+1. For i=1,...,n_s, the G(i) are unchanged from the previous step. The information necessary for generating the remaining G(i) is stored by the previous step in the array V(q) according to

$$\begin{aligned}
V(1) &= g_{n_s 1} \\
V(2) &= g_{n_s 2} \\
&\vdots \\
V(k+2-n_s) &= g_{n_s,k+2-n_s} \\
V(k+3-n_s) &= g_{n_s-1,k+3-n_s} \qquad\qquad (5\text{-}18) \\
V(k+4-n_s) &= g_{n_s-2,k+4-n_s} \\
&\vdots \\
V(k) &= g_{2k}
\end{aligned}$$

For this step, V(q) is updated according to

$$V(q) \leftarrow V(q) - \alpha_{n_s}(n+1)V(q+1) \quad q=1,2,\ldots,k+1-n_s \tag{5-19}$$

The G(i) are generated using a working vector W(q) which is initialized by

$$W(q) \leftarrow V(q) \qquad q = 1,\ 2,\ \ldots,\ k+1-n_s \tag{5-20}$$

Then $G(n_s+1)=W(1)$, and the remaining G(i) are obtained by calculating in the order $i = n_s+2, \ldots, k+1$:

$$\begin{aligned} &W(q) \leftarrow W(q) - \alpha_{i-1}(n+1)W(q+1) \quad q = 1,\ 2,\ \ldots,\ k+2-i \\ &G(i) \leftarrow W(1) \end{aligned} \tag{5-21}$$

If the order is lowered, no special action is required in computing the coefficients, since the only effect is that some of the stored values are not used. If the order is raised, $k=k_{old}+1$, V must be updated according to

$$V(k) \leftarrow \frac{1}{k(k+1)}$$

$$V(k-j) \leftarrow V(k-j) - \alpha_{j+1}(n+1)V(k-j+1) \qquad j=1,2,\ldots,n_s-2$$

after which the computation of G(i) proceeds as before starting with (5-19).

If the stepsize is changed, then $n_s=1$ and all the coefficients must be recomputed. In this case, V and W are initialized as

$$W(q) \leftarrow V(q) \leftarrow \frac{1}{q(q+1)} \qquad q = 1,\ 2,\ \ldots,\ k$$

and the G(i) are then computed according to (5-21).

The PECE method of order k produces the k+1 order solution y_{n+1} given by (5-17). If the corrector formula were of order k rather than order k+1, then the k order solution

$$y_{n+1}(k) = p_{n+1} + h_{n+1}g_{k1}\phi^{p}_{k+1}(n+1)$$

would be obtained. STEP uses the difference between these two solutions as an estimate of the local error in the step:

$$\mathrm{ERR} = \left|h_{n+1}(g_{k+1,1}-g_{k1})\phi^{p}_{k+1}(n+1)\right| \tag{5-22}$$

The step is accepted or rejected depending on whether or not ERR≤EPS, where EPS is the local error tolerance specified by the user of the code. This test does not require the final function evaluation, so this evaluation is not done if the step fails.

Regardless of whether the step is successful, the following quantities are used to test whether the order should be lowered:

$$\begin{aligned} \text{ERKM2} &= |h\gamma^*_{k-2}\sigma_{k-1}(n+1)\phi^P_{k-1}(n+1)| \\ \text{ERKM1} &= |h\gamma^*_{k-1}\sigma_k(n+1)\phi^P_k(n+1)| \\ \text{ERK} &= |h\gamma^*_k\sigma_{k+1}(n+1)\phi^P_{k+1}(n+1)| \end{aligned} \tag{5-23}$$

where the σ's are computed along with the other coefficients by using

$$\begin{aligned} \sigma_i(n+1) &= 1 & i&=1,2,\ldots,n_s+1 \\ \sigma_i(n+1) &= (i-1)\alpha_{i-1}(n+1)\sigma_{i-1}(n+1) & i&=n_s+2,\ldots,k+1 \end{aligned} \tag{5-24}$$

ERKM2, ERKM1, and ERK are estimates of the errors which would occur at, respectively, orders k-2, k-1, and k, if a constant stepsize were being used. The order is lowered to k-1 if

$$\begin{aligned} &k > 2 \text{ and } \max(\text{ERKM1},\text{ERKM2}) \le \text{ERK} \\ &k = 2 \text{ and } \text{ERKM1} \le 0.5\ \text{ERK} \end{aligned} \tag{5-25}$$

A test for raising the order is made only if (1) the step is successful, (2) the stepsize is actually constant ($n_s \ge k+1$), and (3) it has not already been decided to lower the order. If these three conditions are fulfilled, the quantity

$$\text{ERKP1} = |h\gamma^*_{k+1}\phi_{k+2}(n+1)| \tag{5-26}$$

where

$$\phi_{k+2}(n+1) = \phi_{k+1}(n+1) - \phi_{k+1}(n)$$

is calculated as an estimate of the error at order k+1. The order is raised if

$$\begin{aligned} &1 < k < 12 \quad \text{and} \quad \text{ERKP1} < \text{ERK} \\ &k = 1 \quad \text{and} \quad \text{ERKP1} < 0.5\ \text{ERK} \end{aligned} \tag{5-27}$$

After the order is selected, it is renamed k, and the corresponding error estimate for constant stepsize is renamed ERK. To economize in the evaluation of the coefficients, the stepsize selection procedure is biased in favor of using a constant stepsize. Stepsize increase is made only by doubling, which is performed only if

$$0.5 \text{ EPS} \geq 2^{k+1} \text{ ERK}$$

If the stepsize cannot be doubled, the current stepsize is retained provided

$$0.5 \text{ EPS} \geq \text{ERK}$$

If this condition is not satisfied, the stepsize is reduced by the factor

$$r = \left(\frac{0.5 \text{ EPS}}{\text{ERK}}\right)^{\frac{1}{k+1}}$$

except that r is not permitted to be greater than 0.9 nor smaller than 0.5.

By aiming for an error of 0.5 EPS rather than EPS, the stepsize is chosen conservatively so as to prevent unnecessary step failures.

To start the integration, the method of order 1, which requires no memorized values, is used. The user provides an upper bound $h_{\max}$ for the size of the first step, and STEP computes a starting stepsize from

$$h = \min\left(h_{\max}, \frac{1}{4}\left|\frac{0.5 \text{ EPS}}{f(t_0, y_0)}\right|^{\frac{1}{2}}\right)$$

During the first phase of the integration, signaled in the code by the logical variable PHASE1 set .TRUE., the stepsize is doubled and the order is raised after every step. The purpose of this device is to reach the stepsize and order appropriate for the problem as rapidly as possible. This first phase is terminated when there is a step failure, when (5-25) says to lower the order, or when the maximum order of 12 is reached.

After leaving the starting phase, STEP follows the previously stated rules for order and stepsize selection with one exception: if three successive failures occur in the same step, all memorized values are discarded and the order is set to 1. This is done because such a sequence of failures indicates a drastic change in the character of the solution, so that the use of higher order methods cannot be justified.

The interpolating polynomial $P^*_{k+1,n}$ in (5-9) can be used not only to approximate the solution at t_{n+1}, but also at any t near t_{n+1} according to

$$y(t) \simeq y_n + \int_{t_n}^{t} P^*_{k+1,n}(\tau)\, d\tau$$

This allows the solution to be obtained at points which are not mesh points if the integration, so that it is not necessary of adjust the stepsize to make mesh points and output points coincide, as must be done for Runge-Kutta methods. The subroutine SINTRP, when used in conjunction with STEP, performs this interpolation of the solution.

Let t_{out} be an output point such that

$$t_n \le t_{out} \le t_{n+1}$$

Then the solution and its derivative at t_{out} are calculated by SINTRP as

$$y_{out} = y_{n+1} + h_I \sum_{i=1}^{k+1} g_{i1}^I \phi_i(n+1)$$

$$y'_{out} = \sum_{i=1}^{k+1} \rho_i \phi_i(n+1)$$

where the coefficients are generated from

$$g_{1q}^I = \frac{1}{q}$$

$$g_{iq}^I = \Gamma_{i-1} g_{i-1,q}^I - \eta_{i-1} g_{i-1,q+1}^I \qquad i \ge 2$$

and

$$\rho_1 = 1$$

$$\rho_i = \rho_{i-1}\Gamma_{i-1} \qquad i \ge 2$$

where

$$\Gamma_1 = \eta_1$$

$$\Gamma_i = \eta_i + \frac{\psi_{i-1}(n+1)}{\psi_i(n+1)} \qquad i \ge 2$$

$$\eta_i = \frac{h_I}{\psi_i(n+1)} \qquad i \ge 1$$

$$h_I = t_{out} - t_{n+1}$$

By interpreting the foregoing formulae in terms of vectors, their application to a system of n equations (2-2) is straightforward. The only modification required is in the error estimates (5-22), (5-23) and (5-26), which are computed in STEP as

$$\text{error} = \max_{L=1,\dots,n} \left| \frac{\text{error}(L)}{\text{WT}(L)} \right|$$

where error(L) is the error estimated for the L-th component of the solution, and WT(L) is a vector of nonzero weights specified by the user of the code, as described below in §7.7.

STEP is functionally organized into five blocks:

Block 0: Checks that the stepsize and error tolerance are not too small for the machine precision, and performs the initialization necessary for the first step of the integration.

Block 1: Computes the α, β, σ, ψ, and g coefficients needed for the step.

Block 2: Computes the predicted solution, evaluates the derivatives using the predicted solution, tests whether the order should be lowered, and tests whether the step is successful. If the step succeeds, control is passed to Block 4. If the step fails, control passes to Block 3.

Block 3: Restores variables for repeating the failed step, sets the order selected in Block 2, halves the stepsize, and transfers control to Block 1 to repeat the step.

Block 4: Corrects the predicted solution, evaluates the derivatives, updates the divided differences, sets the order and stepsize appropriate for the next step, and returns to the calling program.

As a means of controlling propagated roundoff error, STEP incorporates an alternate route of computation in the DO loops terminating at statements 25, 235, and 405. The operation of this device is described in Chapter 9 of Shampine(1975). Since roundoff control is not important except near the limits of the machine precision, these loops are not used if the error tolerance is greater than 100 times limiting precision on the first call to STEP.

The code ADAM is an interfacing routine designed to provide the user with an access to the Adams methods which is similar to that provided by the Runge-Kutta code RKFS. The test for stiffness which is used in ADAM is that described in Shampine(1975): the problem is labeled as stiff if a sequence of 50 consecutive steps are taken with order less than or equal to 4. Since the lower order PECE methods have larger stability regions than higher order methods, they will tend to be selected when the problem is stiff, as their larger stability regions will allow larger steps to be taken. While this procedure will also label as stiff some nonstiff problems which happen to be most efficiently integrated by low order methods, most false indications of stiffness are avoided because a finding of stiffness is not reported to the user unless the code also finds that too much work is required to perform the integration.

SECTION 6

STIFF METHODS

Among methods suitable for solving stiff problems, the backward differentiation methods of orders 1 through 6 are the best understood and most widely used. Their use was popularized through a code written by Gear(1971,1971a), for which reason they are sometimes referred to as Gear's method.

The solution of (2-5) at t_i satisfies

$$\left.\frac{dy(t)}{dt}\right|_{t_i} = f(t_i,y(t_i)) \tag{6-1}$$

The backward differentiation method of order k approximates (6-1) by replacing $y(t_i)$ with y_i, and by replacing $y(t)$ with the (unique) k degree polynomial $Q_{ki}(t)$ which interpolates to y_i and to the computed solution values at the k preceding points. This yields an implicit integration method in which y_i is obtained by solving the (generally nonlinear) algebraic equation

$$Q'_{ki}(t_i) = f(t_i,y_i) \tag{6-2}$$

For k=1, this is the backward Euler method (2-6b).

Suppose that the solution has been obtained up to t_{i-1}, so that $Q_{k,i-1}(t)$ is known. When the solution is advanced to t_i, the new polynomial $Q_{k,i}(t)$ is obtained, which satisfies (6-2),

$$Q'_{ki}(t_i) = f(t_i,Q_{ki}(t_i)) \tag{6-3}$$

and which interpolates to y_{i-1}, y_{i-2}, ..., y_{i-k},

$$Q_{ki}(t_{i-\ell}) = y_{i-\ell} = Q_{k,i-1}(t_{i-\ell}) \tag{6-4}$$

$$\ell = 1, 2, \ldots, k$$

Define the k degree polynomial $P_{ki}(t)$ by

$$P_{ki}(t) = Q_{ki}(t) - Q_{k,i-1}(t) \tag{6-5}$$

From (6-4), P_{ki} must satisfy

$$P_{ki}(t_{i-\ell}) = 0 \qquad \ell = 1, 2, \ldots, k \tag{6-6}$$

The condition (6-6) completely determines P_{ki} up to a constant factor. To simplify the discussion, assume that the stepsize has been constant for at least k steps. Then $P_{ki}(t)$ may be written as

$$P_{ki}(t) = e_i S_k\left(\frac{t-t_i}{h}\right)$$

$$\text{where} \quad S_k(x) = -\frac{(x+1)(x+2)\cdots(x+k)}{k!\left(1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{k}\right)} \tag{6-7}$$

The coefficients of $S_k(x) = a_1 + a_2x + a_3x^2 + \cdots + a_{k+1}x^k$ are given in the following table,

k =	1	2	3	4	5	6
$-a_1$	1	$\frac{2}{3}$	$\frac{6}{11}$	$\frac{12}{25}$	$\frac{60}{137}$	$\frac{20}{49}$
$-a_2$	1	1	1	1	1	1
$-a_3$		$\frac{1}{3}$	$\frac{6}{11}$	$\frac{7}{10}$	$\frac{225}{274}$	$\frac{58}{63}$
$-a_4$			$\frac{1}{11}$	$\frac{1}{5}$	$\frac{85}{274}$	$\frac{5}{12}$
$-a_5$				$\frac{1}{50}$	$\frac{15}{274}$	$\frac{25}{252}$
$-a_6$					$\frac{1}{274}$	$\frac{1}{84}$
$-a_7$						$\frac{1}{1764}$

Rewriting (6-5) as

$$Q_{ki}(t) = Q_{k,i-1}(t) + e_i S_k\left(\frac{t-t_i}{h}\right) \tag{6-8}$$

and substituting this into (6-3) gives, after a little rearrangement,

$$e_i = hy'_{i(0)} - hf(t_i, y_{i(0)} + a_1 e_i) \tag{6-9}$$

where $y_{i(0)} = Q_{k,i-1}(t_i)$ and $y'_{i(0)} = Q'_{k,i-1}(t_i)$

The algebraic equation (6-9) may be solved for e_i by a Newton iteration starting with $e_{i(0)}=0$ and proceeding according to

$$\begin{aligned}
c_{i(m+1)} &= \left(1+ha_1\frac{\partial f}{\partial y}\right)^{-1} \cdot (hy'_{i(m)} - hf(t_i, y_{i(m)})) \\
y_{i(m+1)} &= y_{i(m)} + a_1 c_{i(m+1)} \\
hy'_{i(m+1)} &= hy'_{i(m)} - c_{i(m+1)} \\
e_{i(m+1)} &= e_{i(m)} + c_{i(m+1)}
\end{aligned} \tag{6-10}$$

where $\partial f/\partial y$ is evaluated at $(t_i, y_{i(m)})$. If h is sufficiently small, this iteration will always converge,

$$\begin{aligned}
c_{i(\ell)} &\to 0 \\
y_{i(\ell)} &\to y_i = Q_{ki}(t_i) \\
hy'_{i(\ell)} &\to hy'_i = Q'_{ki}(t_i) \\
e_{i(\ell)} &\to e_i
\end{aligned} \tag{6-11}$$

and the polynomial $Q_{ki}(t)$ is obtained from (6-8).

For k = 1, 2, ..., 6, the stability region of the backward differentiation method of of order k includes that portion of the left half of the complex plane which lies between the two lines passing through the origin and making angles $\pm\theta_k$ with the negative real axis, where

$\theta_1 = 90^\circ$ $\quad\theta_4 = 73.5^\circ$

$\theta_2 = 90^\circ$ $\quad\theta_5 = 51.8^\circ$

$\theta_3 = 86.0^\circ$ $\quad\theta_6 = 17.2^\circ$

Plots of the full stability regions are shown in Gear(1971). If the problem being integrated is such that all the eigenvalues λ of the Jacobian matrix fall within the wedge-shaped region specified by θ_k, then $h\lambda$ will fall within this region for all h, and the backward differentiation method of order k will not limit the stepsize for reasons of stability. It is this feature which makes the backward differentiation methods suitable for solving stiff problems, although the occurrence of an eigenvalue close to the imaginary axis may preclude use of the higher order methods.

The truncation error of the backward differentiation method of order k is given by

$$\tau_i = \frac{1}{k+1} h^{k+1} y^{(k+1)}(\xi) \qquad (6\text{-}12)$$

where ξ is some value in the interval between t_{i-1} and t_i. Thus, for $k>1$, the backward differentiation method of order k is less accurate than the same order Adams-Moulton method. For this reason, the backward differentiation methods are more efficient only for stiff problems, for which Adams methods must limit the stepsize for reasons of stability rather than for reasons of accuracy.

Let the interpolating polynomial $Q_{k\ell}(t)$ be represented by the vector $\vec{b}(\ell)$ of its coefficients when it is expressed in terms of powers of $(t-t_\ell)/h$,

$$Q_{k\ell}(t) = b_1(\ell) + b_2(\ell)\cdot s + b_3(\ell)\cdot s^2 + \cdots + b_{k+1}(\ell)\cdot s^k$$

$$\text{where} \quad s = \frac{t-t_\ell}{h}$$

These coefficients may be written as

$$b_{j+1}(\ell) = \frac{h^j \cdot y_\ell^{(j)}}{j!} \qquad (6\text{-}13)$$

where $y_\ell^{(j)}$ approximates the j-th derivative of the solution at t_ℓ. The polynomial S_k is similarly represented by the vector $\vec{a} = (a_1, a_2, \ldots, a_{k+1})$ of its coefficients. With this notation, (6-8) may be written as

$$\vec{b}(i) = T\cdot\vec{b}(i-1) + e_i\vec{a} \qquad (6\text{-}14)$$

where T is the Pascal triangle matrix

$$T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\ & 1 & 2 & 3 & 4 & \cdots & k \\ & & 1 & 3 & 6 & \cdots & \cdot \\ & & & 1 & 4 & \cdots & \cdot \\ & 0 & & & 1 & \cdots & \cdot \\ & & & & & \ddots & \cdot \end{bmatrix}$$

The formulation (6-13) and (6-14) suggest a convenient means of altering the stepsize: to make the next step with stepsize h_{new}, it is only necessary to scale the components of $\vec{b}(i)$ according to

$$b_j(i) \leftarrow \left(\frac{h_{\text{new}}}{h}\right)^{j-1} \cdot b_j(i) \qquad j = 1, \ldots, k+1 \tag{6-15}$$

Since (6-14) was derived under the assumption of a constant stepsize, this procedure can be justified only if the stepsize changes are neither too large nor too frequent. Accordingly, in the code GEAR,

1.) Stepsize increases are limited to a factor no more than 2, and decreases to a factor not smaller than 1/10.

2.) No stepsize or order change is made until there has been a sequence of k+1 successful steps since the last change, unless a step failure necessitates a reduction in stepsize and/or order.

The truncation error (6-12) may be written as

$$\tau_i = \frac{1}{k+1} h^{k+1} y^{(k+1)}(t_i) + \text{higher order terms} \tag{6-16}$$

By ignoring the higher order terms in (6-16), the magnitude of the local error is estimated as

$$\eta = \left| \frac{1}{k+1} h^{k+1} y_i^{(k+1)} \right| \tag{6-17}$$

$$\text{with} \quad y_i^{(k+1)} = \frac{y_i^{(k)} - y_{i-1}^{(k)}}{h}$$

$$= \frac{k!}{h^{k+1}} (b_{k+1}(i) - b_{k+1}(i-1))$$

$$= \frac{k!}{h^{k+1}} e_i \, a_{k+1} \tag{6-18}$$

where the last equality follows from the observation that multiplication by T in (6-14) does not alter the value of $b_{k+1}(i-1)$. In GEAR, the constants $(k+1)/(k!a_{k+1})$ are stored in the array ERTST, and the quantity

$$D = \left| \frac{\eta}{\text{EPS}} \right| = \left| \frac{e_i}{\text{ERTST(k)*EPS}} \right| \tag{6-19}$$

is calculated, where EPS is the local error tolerance specified by the user of the code. Thus, the step succeeds if $D \leq 1$ and fails if $D > 1$.

If the rule (2) above allows a change in stepsize or order, the optimum stepsize for continuing integration with the present order k is estimated as h/R, where

$$R = 1.2(2D)^{1/(k+1)} \tag{6-20}$$

The factor 2 is equivalent to aiming for an error of 0.5 EPS, in order to prevent unnecessary step failures, while the overall factor 1.2 provides an allowance for the approximations made in estimating the local error by (6-17).

The stepsize appropriate for order k-1 is estimated as h/RDWN, where

$$\begin{aligned} \mathrm{RDWN} &= 1.3 \left| \frac{2 \frac{1}{k} h^k y_i^{(k)}}{\mathrm{EPS}} \right|^{1/k} \\ &= 1.3 \left| \frac{2\ (k-1)!\ b_{k+1}(i)}{\mathrm{EPS}} \right|^{1/k} \\ &= 1.3 \left| \frac{2\ b_{k+1}(i)}{\mathrm{ERDWN}(k)*\mathrm{EPS}} \right|^{1/k} \end{aligned} \tag{6-21}$$

where the array ERDWN contains the quantities $1/(k-1)!$.

Raising the order is considered only if the step was successful. Then, the stepsize appropriate for order k+1 is estimated as h/RUP, where

$$\mathrm{RUP} = 1.4 \left| \frac{2 \frac{1}{k+2} h^{k+2} y_i^{(k+2)}}{\mathrm{EPS}} \right|^{1/(k+2)}$$

$$\begin{aligned} \text{with} \quad y_i^{(k+2)} &= \frac{y_i^{(k+1)} - y_{i-1}^{(k+1)}}{h} \\ &= \frac{k!\ a_{k+1}}{h^{k+2}} (e_i - e_{i-1}) \end{aligned} \tag{6-22}$$

where the last equality follows from use of (6-18). The constants $(k+2)/(k!a_{k+1})$ are stored in the array ERUP, and RUP is calculated by

$$\text{RUP} = 1.4 \left| \frac{2\,(e_i - e_{i-1})}{\text{ERUP(k)*EPS}} \right|^{1/(k+2)} \qquad (6\text{-}23)$$

The order for the next step is set to that which corresponds to the smallest of the values R, RDWN, and RUP. The factors 1.3 and 1.4 in (6-21) and (6-23) bias the choice, first, in favor of retaining the same order, and, second, in favor of lowering the order. Thus, the order is raised only if there appears to be a substantial advantage in doing so.

If the order is unchanged, the stepsize is not changed unless $|R-0.95| \leq 0.06$. This avoids the need for reevaluating the Jacobian, as described below, in response to an insignificant stepsize change. Otherwise, the stepsize is changed to h/R, subject to the constraint of rule (1) above.

If the order is changed to k-1 or k+1, the Jacobian must be reevaluated anyway, and the new stepsize is set to h/RDWN or h/RUP, or as limited by rule (1). The only exception to this is when the order is lowered after a step failure, in which case the new stepsize is not permitted to be larger than $h/1.1$.

Lowering the order is accomplished simply by ignoring $b_{k+1}(i)$ in starting the next step. When the order is raised, the quantity

$$\frac{h^{k+1}\, y_i^{(k+1)}}{(k+1)!} = b_{k+2}(i) = \frac{e_i a_{k+1}}{k+1}$$

as given by (6-18), is appended to $\vec{b}(i)$ for use in starting the next step.

The integration is started by using the method of order 1, which requires no memorized values. The initial stepsize is chosen in the same manner as for the Adams code STEP, as described in Section 5.

Applying the formulae given above to a system of n equations (2-2) is a fairly straightforward matter of interpreting these formulae as vector equations, with a separate component for each of the n components of the solution. Since (6-19), (6-21), and (6-23) must produce single numbers, the absolute value signs in these equations are interpreted as vector norms, according to

$$|z| \rightarrow \|\vec{z}\| = \max_{\ell=1,\ldots,n} \left| \frac{z_\ell}{\text{WT}(\ell)} \right|$$

where WT is a vector of nonzero weights specified by the user of the code, as described in §7.7.

The most significant complication occurs in (6-10), where the quantity $(1+ha_1\partial f/\partial y)$ must be replaced by $(I+ha_1 J\vec{f})$, where $J\vec{f}$ is the Jacobian matrix (2-3) and I is the nxn identity matrix. In GEAR, this matrix is not actually inverted. Instead, the equation

$$(I + ha_1 J\vec{f})\ \vec{c}_{i(m+1)} = h\vec{y}_{i(m)} - h\vec{f}(t_i, \vec{y}_{i(m)}) \qquad (6\text{-}24)$$

is solved numerically for $\vec{c}_{i(m+1)}$. The most expensive part of this solution is performed by the subroutine DECOMP, which is taken from Forsythe(1967). DECOMP factors the matrix on the left hand side of (6-24) into a product of upper and lower triangular matrices. After this is done, it is easy and inexpensive to solve the equation. DECOMP performs a number of operations which increases as n^3, so that this may be the most expensive part of the entire integration when the number of equations is large. To economize on this expense, this matrix is reevaluated only when the iteration (6-10) will not converge, or when the stepsize or order are changed. Up to three iterations (6-10) are taken; the iteration is stopped when practical convergence is indicated by

$$\|\vec{c}_{i(m)}\| \leq \frac{\text{EPS}}{2\cdot n\cdot(k+2)}$$

If convergence does not occur in three iterations, the matrix in (6-24) is first reevaluated and the calculation is repeated. If this does not suffice to produce convergence, the stepsize is reduced to $h/4$ for another attempt.

In GEAR, the polynomial coefficients (6-13) are in the array Y(*,*), where the first index denotes the component of the solution and the second denotes the order of the coefficient. GEAR is functionally organized into five blocks:

Block 0: Checks whether the stepsize and error tolerance are too small for the machine precision. Performs necessary initializations the first time GEAR is called.

Block 1: Computes coefficients to correspond to the order of the method, and scales Y(*,*) in accordance with any stepsize change.

Block 2: Advances the solution one step, and tests whether the error criterion is met. If the step succeeds, control is passed to Block 4. If the step fails, control passes to Block 3.

Block 3: Restores values to be used for repeating the failed step. Tests whether to lower the order, reduces the stepsize, and transfers control to Block 1 to repeat the step.

Block 4: Completes the successful step by updating those components of Y(*,*) which were not computed in Block 2. Sets order and stepsize appropriate for the next step, and returns control to the calling program.

Subroutine GINTRP, when used in conjunction with GEAR, interpolates the solution in order to provide output at points which are not mesh points of the integration. It is used in the same way as the subroutine SINTRP for the Adams method code STEP, as described in Section 5.

The code STIFF is an interfacing routine designed to provide the user with an access to the methods of GEAR which is similar in format to the Runge-Kutta code RKFS and the Adams method code ADAM.

SECTION 7

USING THE CODES

§7.1 *Implementing the Codes*

The three packages of subroutines, Runge-Kutta Programs, Adams Method Programs, and Stiff Method Programs, are independent of one another and may be implemented separately. The actual integration is performed by one of the subroutines RKFS, STEP, and GEAR. The remaining programs are interfacing routines designed to give the user a more convenient, though less flexible, mode of access. The relationships among the various subroutines are shown in Figure 7-1. The boxes connected by dotted lines represent named COMMON blocks used to pass parameter values between the subroutines. The subroutines denoted by *OUTP*, *FCT*, and *FDER* are not included in the packages, but are subroutines supplied by the user. Complete listings of the programs are contained in Appendices A, B, and C.

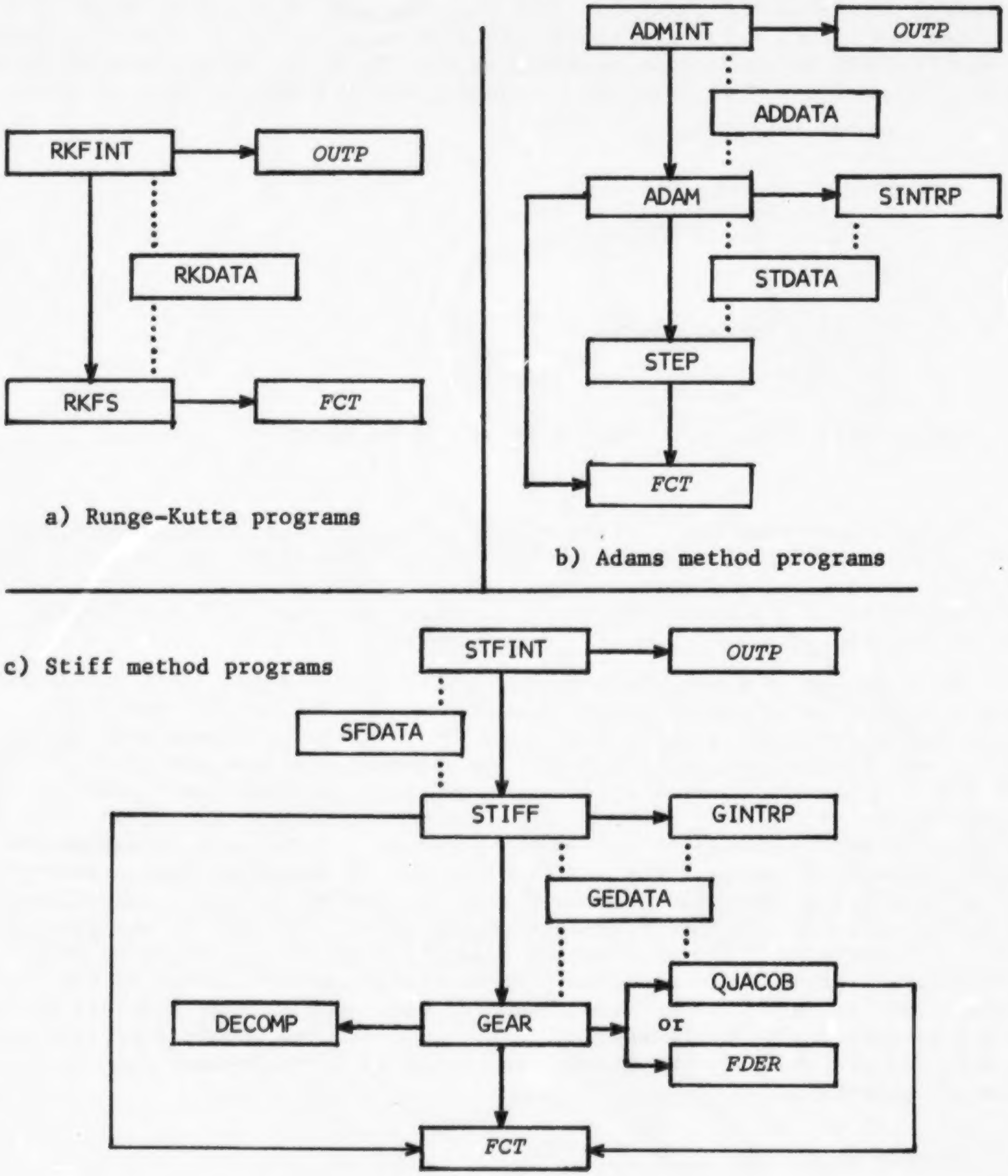
All the subroutines are written to conform to the specifications of ANSI FORTRAN IV. The only known departure from this standard is in the form of those DATA statements in STEP and GEAR which involve arrays. However, the form of these DATA statements is believed to be acceptable to all current implementations of FORTRAN IV.

The subroutines RKFS, ADAM, STEP, STFINT, GEAR, and QJACOB contain machine dependent constants which are used to prevent attempts to compute beyond the limits set by the machine precision. Before the programs are compiled, the DATA statements which set these constants must be modified to provide values appropriate for the computer being used. To avoid logical conflicts when integrating at high precision, it is essential that these constants be set identically for all subroutines in the same package. The recommended values are

$$\begin{aligned} \text{FOURU} &= 4u \\ \text{U26} &= 26u \\ \text{REMIN} &= \text{larger of } 4u \text{ and } 10^{-12} \end{aligned}$$

where u is the computer unit roundoff error, the smallest positive value such that $1.+u$ is greater than 1 in floating point arithmetic. It is given by

$$\begin{aligned} u &= \beta^{1-s} \quad \text{for truncated arithmetic} \\ &\text{or} \qquad\qquad\qquad\qquad\qquad\qquad (7\text{-}1) \\ u &= \tfrac{1}{2}\beta^{1-s} \quad \text{for rounded arithmetic} \end{aligned}$$


RKFINT
OUTP
RKDATA
RKFS
FCT
ADMINT
OUTP
ADDATA
ADAM
SINTRP
STDATA
STEP
FCT
STFINT
OUTP
SFDATA
STIFF
GINTRP
GEDATA
QJACOB
or
FDER
DECOMP
GEAR
FCT

a) Runge-Kutta programs

b) Adams method programs

c) Stiff method programs

Figure 7-1. Organization of the Subroutine Packages.
Arrow direction is from calling program to called program. Those blocks connected only by dotted lines represent COMMON blocks. *OUTP*, *FCT*, and *FDER* represent subroutines supplied by the user.

where β is the base of the floating point representation, and s is the number of base β digits in the mantissa. (If $\beta=2$, the most significant digit of the mantissa may be not actually stored, since it is always 1 in normalized form. It should still be counted in determining s.) If u is unknown, a sufficiently accurate value may be obtained by running the following simple program.

```
      DOUBLE PRECISION DU
      U=1.
   10 U=U/2.
      IF ((1.+U).GT.1.) GO TO 10
      U=2.*U
      DU=1.D0
   20 DU=DU/2.D0
      IF ((1.D0+DU).GT.1.D0) GO TO 20
      DU=2.D0*DU
      WRITE(6,100) U,DU
      STOP
  100 FORMAT (9H S.P. U =,E12.4/9H D.P. U =,E12.4)
      END
```

Variables in common blocks must be ordered so as to force proper alignment between the variables and word boundaries. The order used here is real variables first, integer variables second, and logical variables last. With most installations, this should produce correct alignment for both single and double precision versions of the programs.

The programs as given are written in single precision. This should be adequate for most purposes, except possibly on the IBM 360/370 machines, whose short precision is quite poor. In the programs, all constants whose values are critical are representable exactly in the single precision of the IBM 360/370, and hence are exact for all machines on which these programs are likely to be implemented. Since all arithmetic operations have been coded so as to force double precision arithmetic whenever all the real variables are double precision, the programs may be converted to double precision simply by declaring all real variables double precision, and by replacing all library functions with their double precision counterparts. In each of the programs there are several contiguous lines of comments which will accomplish this conversion if they are turned into program statements by removal of the C in the first column. The machine dependent constants must also be set to the values correct for double precision. Any conversion to double precision must be done for *all* the programs in the same package, for else they cannot interact properly.

§7.2 *Specifying the Equation*

The user specifies the differential equations to be solved by supplying a subroutine, denoted by *FCT* in Figure 7-1, which evaluates the derivatives $\vec{f}(t,\vec{y})$ in (2-2). A call to *FCT* will be referred to as a *function evaluation*. *FCT* must have the argument list (T,Y,YP), where T is a real variable and Y and YP are real arrays each of dimension n, where n is the number of equations. When called, *FCT* must compute $\vec{f}(t,\vec{y})$ for $(t,\vec{y})$=(T,Y), and place these derivative values in YP. *FCT* must not alter the values of T and Y.

Frequently, $\vec{f}$ will depend upon a number of parameters which the user wishes to vary in order to observe the effect upon the solution. This is most easily accomplished by coding these parameters as variables in *FCT*, and then passing their values from the main program through a COMMON block.

As an example, the *FCT* subroutine for the simple ecosystem model (1-2) might be coded as follows:

```
SUBROUTINE SIMPLE(T,Y,YP)
DIMENSION Y(4),YP(4)
COMMON A14,A21,A32,A42,A43
F14=A14*Y(4)
F21=A21*Y(1)
F32=A32*Y(2)
F42=A42*Y(2)
F43=A43*Y(3)
YP(1)=F14-F21
YP(2)=F21-F32-F42
YP(3)=F32-F43
YP(4)=F42+F43-F14
RETURN
END
```

During the integration, many function evaluations are performed. To make the integration as inexpensive as possible, *FCT* must be carefully coded to eliminate redundant and unnecessary operations. This is particularly important if $\vec{f}(t,\vec{y})$ is expensive to evaluate, since function evaluations will then contribute a large portion of the integration cost.

The stiff method programs additionally require a subroutine, denoted by *FDER* in Figure 7-1(c), to evaluate the Jacobian matrix $J\vec{f}(t,\vec{y})$ defined by (2-3). For *FDER*, the user may specify the subroutine QJACOB, which employs *FCT* to compute the Jacobian by numerical differencing. However, unless the form of the Jacobian is very complex, it is preferable to supply an *FDER* which evaluates the Jacobian directly. *FDER* must have the argument list (T,Y,W,YP,DER), where Y and YP are arrays of dimension n, and W is a two-dimensional array of dimensions (n,n). DER is a subroutine name which is included in the argument list only to accommodate the requirements of QJACOB: when *FDER* is called by GEAR, DER is the subroutine which the user has supplied for *FCT*.

FDER must compute all the partial derivatives of $\vec{f}(t,\vec{y})$ at $(t,\vec{y})$=(T,Y) and place them in W according to

$$W(I,J) \leftarrow \frac{\partial f_I}{\partial y_J} \qquad \begin{array}{l} I = 1, 2, \ldots, n \\ J = 1, 2, \ldots, n \end{array}$$

When *FDER* is called by GEAR, YP contains the derivatives $\vec{f}(t,\vec{y})$ at (T,Y). For some problems, these values may be useful in simplifying the evaluation of the Jacobian. *FDER* must not alter the values of T, Y, and YP.

For the simple ecosystem model (1-2). the Jacobian is constant, and *FDER* might be coded as follows:

```
SUBROUTINE SIMPJ(T,Y,W,YP,DER)
DIMENSION Y(4),YP(4),W(4,4)
COMMON A14,A21,A32,A42,A43
W(1,1)=-A21
W(2,1)=A21
W(3,1)=0
W(4,1)=0
W(1,2)=0
W(2,2)=-A32-A42
W(3,2)=A32
W(4,2)=A42
W(1,3)=0
W(2,3)=0
W(3,3)=-A43
W(4,3)=A43
W(1,4)=A14
W(2,4)=0
W(3,4)=0
W(4,4)=-A14
RETURN
END
```

§7.3 *The Local Error Tolerance*

Except for direct use of STEP or GEAR, whose requirements are described in §7.7 below, the user specifies the local error test by supplying nonnegative values for two parameters: a relative error tolerance E_r and an absolute error tolerance E_a. In the programs, these parameters are denoted by RELERR and ABSERR, respectively. If $y(\ell)$ is the current value of the ℓth component of the solution, the local error tolerance for that component is computed as

$$E(\ell) = E_r|y(\ell)| + E_a \qquad (7\text{-}2)$$

and the codes require that

$$\eta(\ell) \leq E(\ell) \qquad \ell = 1, 2, \ldots, n \qquad (7\text{-}3)$$

for a successful step, where $\eta(\ell)$ is the magnitude of the estimated local error in the ℓth component of the solution.

At least one of E_r and E_a must be nonzero, for else it will not be possible to satisfy (7-3). For a pure relative error test (E_a=0 and E_r>0), $E(\ell)$ will vanish if $y(\ell)$=0. The Runge-Kutta programs compute $|y(\ell)|$ as the average magnitude of the solution at the beginning and end of the step, so that this difficulty is unlikely to arise. However, the Adams and stiff method programs use for $y(\ell)$ the solution value at the beginning of the step, so that a pure relative error test cannot be used for starting a solution which has any component with vanishing initial value.

To avoid attempts at impossible accuracy, the programs also require that

$$E(\ell) \geq 4u|y(\ell)| \qquad \ell = 1, 2, \ldots, n \qquad (7\text{-}4)$$

where u is the computer unit roundoff error (7-1). For the Runge-Kutta programs, (7-4) is automatically satisfied by requiring that RELERR be greater than or equal to REMIN. The Adams and stiff method programs do allow a pure absolute error test (RELERR=0, ABSERR≠0), but a check is made at each step to ensure that (7-4) is satisfied.

In choosing the error tolerances, the user should consider the accuracy of the model which he is investigating. It is pointless to request that errors in the integration be very much smaller than the empirical uncertainties in the model itself. Doing so will simply increase the cost of solution without producing any additional usable information.

The user must be fully aware that his specified error tolerances are used only to control the *local* error in each step of the integration. While it is true for most problems that the resulting global error is the same order of magnitude as the local error tolerances, this cannot be guaranteed, for the reasons discussed in §2.3 above. A reliable estimate of the *global* error may be obtained by repeating the integration with the error tolerances reduced by, say, a factor of 1/10. The difference of the two solutions thus obtained may then be taken as an estimate of the global error in the less accurate solution.

§7.4 *Choice of Method and Choice of Program*

The best method for a particular problem is that which computes the solution most efficiently, i.e., with the least possible expense. Factors which affect the efficiency of the codes are the degree of stiffness of the problem, the expense of the function evaluations, and the stringency of the error tolerances.

The stiff method programs are designed specifically for the solution of stiff problems. Although they will integrate nonstiff problems, they will do so less efficiently than the Runge-Kutta or Adams methods. Normally they are used only when the problem is already known to be stiff, or after stiffness is suspected as the cause of an expensive integration with the Runge-Kutta or Adams programs.

Unless stiffness is fairly severe, the choice is between the Runge-Kutta and Adams methods. The Runge-Kutta methods are best when the function evaluations are cheap and only moderate accuracy is required. The Adams methods are preferred when function evaluations are expensive or when high accuracy is required.

The expense of a function evaluation is judged by the number of operations *per component* required to evaluate $\vec{f}(t,\vec{y})$. If, as in the subroutine SIMPLE in §7.2 above, each component is computed by only a few arithmetic operations, the function evaluation is classified as cheap. On the other hand, if the evaluation of each component requires, on the average, many arithmetic operations, or the evaluation of a transcendental function, the function evaluation must be regarded as expensive. The effect of error tolerance on the relative efficiencies of the Runge-Kutta and Adams methods is illustrated in Tables 8-2 and 8-3 in Section 8.

If, as is often the case, the same problem is to be integrated repeatedly with only minor adjustment of parameters in the definition of $\vec{f}(t,\vec{y})$, it may be worthwhile to perform a trial solution with the different methods in order to determine which is best for the production run.

If the number n of equations is very large, storage requirements may constrain choice of a method. To hold the solution and intermediate results, the programs require the following:

Runge-Kutta programs:	$8n$ floating point locations
Adams method programs:	$20n$ floating point locations
Stiff method programs:	$18n+n^2$ floating point locations and n fixed point (integer) locations

Also, while the amount of computation increases linearly with n for Runge-Kutta and Adams methods, the amount of computation required by the stiff methods has a term which increases as n^3. Thus, when n is large, a moderately stiff problem may be handled more efficiently by the Runge-Kutta or Adams methods than by the stiff method programs. Problems which are both very large and very stiff cannot be integrated by any of these codes. Such problems are tractable only when the Jacobian matrix is sparse (i.e., contains few nonzero elements), in which case a specially coded integration routine may be used to economize on storage requirements and on the cost of matrix inversion.

Once the basic method has been selected, the choice of a particular program depends on the amount of control which the user must have over the integration process. The easiest programs to use are RKFINT, ADMINT, and STFINT. A single call to one of these subroutines calculates the solution over the entire interval of integration, using the mixed relative-absolute error test described in §7.3. Output of the solution at equally spaced points in the interval is accomplished by calls to a user-supplied output subroutine. When an error is encountered, integration is continued if it is possible to take corrective action. If an error cannot be corrected, execution of the program is terminated. Informative and diagnostic messages are written into a data set specified by the user.

Greater flexibility in the choice of output points and in the handling of error conditions may be achieved through direct use of RKFS, ADAM, or STIFF. A call to one of these programs advances the solution to the output point specified by the user, using a mixed relative-absolute error tolerance. Return to the calling program occurs when the output point is reached, or sooner, if an error condition occurs. The complete solution requires a sequence of calls to the integration routine, each call carrying the integration from one output point to the next. Using these programs is considerably more demanding of the user, since all provisions for output and error handling must be written into the calling program.

For Adams and stiff methods, the greatest control over the integration can be achieved by direct use of STEP or GEAR. Each of these subroutines returns control to the calling program after advancing the solution over one step. They also allow specification of a different error test for each component of the solution. For the Runge-Kutta method, similar step-by-step monitoring of the solution can be achieved by using RKFS in single-step mode. In this case, however, the error test is limited to the simple relative--absolute specification.

§7.5 *Using* RKFINT, ADMINT, *and* STFINT

A single call to RKFINT, ADMINT, or STFINT will produce a complete integration of the user's problem. The user specifies the beginning and end of the integration interval by TINIT and TFINAL, and the spacing of output points by TINCR. To provide the output, the program calls a user-supplied output subroutine, denoted by *OUTP* in Figure 7-1, at intervals of TINCR in the independent variable. The *OUTP* subroutine must have an argument list of the for (T,Y,YP), where T is a real variable, and Y and YP are real arrays each of dimension NEQN, where NEQN is the number of equations being integrated. When *OUTP* is called, T contains the current value of the independent variable, Y contains the solution at T, and YP contains the derivatives of the solution at T. (YP is actually the working storage array W described below, but only the first NEQN locations of this array, which contain the derivative values, are likely to be of interest to the user.)

As an example, the following output subroutine may be used to print out the solution and derivative values for the problem specified by the subroutine SIMPLE in §7.2. For this problem, the sum of the components, Y(1)+Y(2)+Y(3)+Y(4), should be constant. As a check against possible coding errors, and as a rough indication of the solution's accuracy, this output routine also computes and prints out this sum.

```
      SUBROUTINE SIMOUT(T,Y,YP)
      DIMENSION Y(4),YP(4)
      SUM=Y(1)+Y(2)+Y(3)+Y(4)
      WRITE(6,10) T,Y,SUM,YP
      RETURN
   10 FORMAT(1H ,F7.2,5G12.4,5X,4G12.4)
      END
```

All information necessary for integration by RKFINT, ADMINT, and STFINT is passed to these programs through their call lists, which are

RKFINT(NEQN,Y,W,TINIT,TFINAL,TINCR,RELERR,ABSERR,IUNIT,*FCT*,*OUTP*)

ADMINT(NEQN,Y,W,TINIT,TFINAL,TINCR,RELERR,ABSERR,IUNIT,*FCT*,*OUTP*)

STFINT(NEQN,Y,W,IW,TINIT,TFINAL,TINCR,RELERR,ABSERR,IUNIT,*FCT*,*FDER*,*OUTP*)

The quantities in the call list have the following meanings:

NEQN	the number of equations
Y	a real array of dimension NEQN, into which the user places the initial values of the solution
W	a real array used for working storage. Its minimum dimensions must be 7*NEQN for RKFINT, 19*NEQN for ADMINT, and 17*NEQN+NEQN**2

	for STFINT
IW	(used by STFINT only) an integer array of dimension NEQN, used for working storage
TINIT	the starting value of the independent variable
TFINAL	the value of the independent variable at the end of the integration interval
TINCR	the spacing between the output points
RELERR	the relative error tolerance for the local error test
ABSERR	the absolute error tolerance for the local error test
IUNIT	the logical unit number of the data set to be used for writing informative and diagnostic messages. If IUNIT is less than or equal to zero, no messages will be written.
FCT	the name of the user-supplied subroutine to be used for calculating the derivatives, as described in §7.2
FDER	(used by STFINT only) the name of the user-supplied subroutine to be used for calculating the Jacobian matrix, as described in §7.2
OUTP	the name of the user-supplied subroutine used to handle output

To use one of these programs, the user must

1. Allocate storage for the arrays Y and W (and also for IW, if STFINT is being used) in a DIMENSION statement.
2. Declare the actual names of *FCT* and *OUTP* (and also of *FDER*, if STFINT is being used) in an EXTERNAL statement, and place those names in the call list.
3. Place the initial values of the solution in Y.
4. Provide appropriate values for NEQN, TINIT, TFINAL, TINCR, RELERR, ABSERR, and IUNIT. Since these quantities are not altered by the programs, they may be specified by writing constants in the call list.

As an example, the following main program uses RKFINT to integrate the simple ecosystem model (1-2) over the interval [0,100] with output at steps of 1 and error tolerances RELERR=ABSERR=10^{-4}.

```
      DIMENSION Y(4),W(28)
      EXTERNAL SIMPLE,SIMOUT
      COMMON A14,A21,A32,A42,A43
C GET VALUES OF PARAMETERS
      READ(5,10) A14,A21,A32,A42,A43
C READ INITIAL VALUES
      READ(5,20) Y
C WRITE TABLE HEADINGS FOR OUTPUT
      WRITE(6,30)
C INTEGRATE.  MESSAGES TO BE WRITTEN TO UNIT 4.
      CALL RKFINT(4,Y,W,0.,100.,1.,1.E-4,1.E-4,4,SIMPLE,SIMOUT)
      STOP
```

```
   10 FORMAT(5F12.2)
   20 FORMAT(4F15.3)
   30 FORMAT(3X,4HTIME,5X,4HY(1),12X,4HY(2),12X,4HY(3),12X,3HSUM,
     1       27X,11HDERIVATIVES)
      END
```

To integrate with ADMINT, it is only necessary to change the DIMENSION statement to

```
      DIMENSION Y(4),W(76)
```

and to replace "RKFINT" with "ADMINT" in the CALL statement.

To integrate with STFINT, the DIMENSION, EXTERNAL, and CALL statements must be changed to

```
      DIMENSION Y(4),W(84),IW(4)

      EXTERNAL SIMPLE,SIMOUT,SIMPJ

      CALL STFINT(4,Y,W,IW,0.,100.,1.,1.E-4,1.E-4,4,SIMPLE,SIMPJ,SIMOUT)
```

Integration may be done in the reverse direction; i.e., TFINAL < TINIT is permitted. In this connection, it should be noted that the reverse integration of a stiff problem is a highly unstable problem, and *vice versa*.

The error messages written into the data set specified by IUNIT are self explanatory. Each message contains the value of the independent variable at which the error occurred, and states what action was taken. As an example, a request for too much accuracy is handled by increasing RELERR and ABSERR to achievable values, printing these new values in the error message, and then continuing integration with the increased tolerances.

The *OUTP* subroutine need not be limited to merely writing out the solution. Depending on the user's needs, it may be written to perform calculations requiring the solution values as input, to prepare the output for graphical display, to compare the solution with expected values, or, in an interactive environment, to force a pause in integration to give the user time to assess the quality of the results already obtained. A limited amount of control over the integration can be achieved by using *OUTP* to manipulate some of the values in the COMMON blocks shown in Figure 7-1. For example, if the problem is known to be very expensive to integrate, error messages for "too much work" may be suppressed by setting the function evaluation counter NFE to a large negative value the first time *OUTP* is called, which is at T = TINIT. NFE is in the COMMON block RKDATA, STDATA, or GEDATA, for Runge--Kutta, Adams, or stiff methods, respectively. When using RKFINT, the test for stiffness may be suppressed by referring to the COMMON block RKDATA and setting KST > 25 the first time *OUTP* is called. This will reduce the overhead of integration slightly, but any report of "too much work" will then give a false indication of stiffness.

§7.6 *Using* RKFS, ADAM, *and* STIFF

If unequally spaced output points are needed, or a different handling of errors is required, either the user may write a modified driver program on the model of RKFINT, ADMINT, or STFINT, or he may use RKFS, ADAM, or STIFF directly. Each of these three programs returns control to the calling program after advancing the solution from one output point to the next.

To start the integration, the user supplies the initial conditions (T,Y), sets TOUT to the value of the first output point, and sets IFLAG=1 to indicate that this is the beginning of the problem. Assuming that the integration is successful, the subroutine returns with T = TOUT, with Y the solution at TOUT, and with IFLAG=2. To continue to the next output point, it is only necessary to set the new value of TOUT and call the subroutine again. This process is continued until the entire solution is generated.

Although RKFS, ADAM, and STIFF are similar in their requirements for use, their call lists are quite different. Full instructions for using each of these programs are contained as comments in the program listings in the Appendices. Values of IFLAG other than 2 are used to inform the user of errors which occur during the integration: the significance of each value of IFLAG is explained in the comments to the appropriate program.

RKFS may be used in *single-step mode* by starting the integration with IFLAG=-1 rather than IFLAG=1. In single-step mode, RKFS returns to the calling program after each step of the integration with IFLAG=-2. Calling RKFS again then advances the integration another step, and so forth. Arrival at TOUT is signaled by IFLAG=2. To continue in single-step mode to a new TOUT then requires the user to change IFLAG from 2 to -2.

For reasons of efficiency, ADAM and STIFF integrate beyond TOUT internally and interpolate the solution to TOUT by using the subroutines SINTRP and GINTRP. For some problems, it may not be possible to integrate beyond TOUT because the derivatives are not defined there. Negative values of IFLAG are used to signal ADAM and STIFF that the integration cannot be carried past TOUT. This option should never be used unless there is a genuine difficulty at TOUT, since any subsequent continuation to a new TOUT then requires restarting the integration with a first order method, so that much efficiency will be lost.

§7.7 *Using* STEP *and* GEAR

STEP and GEAR are the step-by-step integration routines for the Adams and STIFF methods, respectively. Full instructions for their use are given as comments in the program listings in Appendices B and C.

These two codes provide for an error tolerance specification which is much more flexible than the relative-absolute test described in §7.3. In STEP and GEAR, the user supplies a basic error tolerance EPS, and specifies how this tolerance is to be applied to each component of the solution by giving a vector of nonzero weights WT(L), L=1,2,...,NEQN. For example,

```
WT(I) = A
```

specifies an absolute error test of A*EPS for the I-th component of the solution,

```
WT(I) = ABS(Y(I))
```

specifies an error test of EPS relative to the most recent value of Y(I), and

WT(I) ← AMAX1(WT(I),ABS(Y(I)))

specifies an error test of EPS relative to the largest magnitude so far achieved by the I-th component. This last example is effectively a relative error test if the I-th component is increasing, and an absolute error test if the I-th component is decreasing.

Rather than manipulating the stepsize to cause exact arrival at an output point, STEP and GEAR are best used in conjunction with the interpolation routines SINTRP and GINTRP. In general, any alteration of the stepsize automatically chosen by STEP or GEAR will cause inefficient operation of the program, and will, consequently, cause the integration to be more expensive than necessary.

STEP and GEAR do not check for errors in input other than requests for a stepsize or an error tolerance which is too small for the precision of the machine. Thus, the user must be quite cautious when using these programs directly. An examination of the listings of ADAM and STIFF may be a useful aid to understanding the necessary precautions.

§7.8 *Miscellanea*

The COMMON blocks shown in Figure 7-1 are used for two purposes: (1) to transmit values from and to return values to the calling program, and (2) to hold values needed for use on subsequent calls to the subroutine. The use of a COMMON block to transmit arguments to a subroutine is usually more efficient than placing the values in the call list, particularly when the number of variables is large. If, however, the use of these COMMON block is inconvenient, it is a simple matter to modify the programs to place these variables in the call lists.

Since no values are retained internally between calls to the subroutines, these programs are easily used in an overlay structure if care is taken to preserve the COMMON blocks from being overwritten. If such a use is not contemplated, then the programs may be modified to allow internal retention of values. Although this is not standard ANSI FORTRAN, it is a commonly used device which allows call lists and interfacing requirements to be greatly simplified.

It is sometimes necessary to integrate problems (2-2) for which $\vec{f}(t,\vec{y})$ has discontinuities in t. While the integration routines given here can usually pass such discontinuities, they can do so only by locating the point of discontinuity through a succession of step failures. It is more efficient to treat such a problem as two distinct problems: the first problem is integrated up to the point of discontinuity, and the solution there is used as initial condition for the second problem, which provides the solution beyond the discontinuity.

SECTION 8

EXAMPLES

This section presents a few simple examples to illustrate how the relative efficiencies of the codes vary with the characteristics of the problem and with the accuracy requested for the integration. These examples were run on the IBM 370 at the Washington Computer Center, using double precision versions of the codes. The integration routines were compiled with the IBM FORTRAN H compiler, OPT=2. Main programs, function subroutines, and output subroutines were compiled with the FORTRAN G compiler, which is faster but produces less efficient object code. The timings quoted are for integration and output only, and do not include the time needed for compilation and linkage of the programs. These timings should not be regarded as highly accurate, since they are somewhat dependent on the number of tasks being handled by the operating system: differences in the least significant digit quoted are not necessarily meaningful.

The first example illustrates the response of the codes to increasing stiffness by integrating the problem

$$\begin{aligned} y' &= -\lambda(y-t^2) + 2t \\ y(0) &= 0 \end{aligned} \qquad (8\text{-}1)$$

This problem has the exact solution $y(t) = t^2$, regardless of the value of λ. However, the problem becomes increasingly stiff for increasingly large positive values of λ. Table 8-1 shows the performance of the codes RKFINT, ADMINT, and STFINT in integrating this problem on the interval [0,50] with output at steps of 1, and error tolerances of RELERR=ABSERR=10^{-5}. The error quoted is the maximum relative error which was found at the output points $t = 1, 2, \ldots, 50$.

In this example, the behavior of the errors is unusual because of the simple form of the solution. For $\lambda = 0$, the solution curves of the differential equation in (8-1) are of the form $u(t) = t^2 + \text{constant}$. An integration method of order greater than or equal to 2 can follow these curves exactly. Since RKFINT uses a method of order 5, and ADMINT uses a method of order at least 2 (because of local extrapolation), the errors in these cases come only from computer roundoff errors. STFINT, however, starts with a method of order 1, which is not exact for this problem, so that the error tolerance permits an error of about 10^{-5}. For $\lambda > 0$, the stability of the problem (8-1) causes the starting error in STFINT to be damped out after the order of the method is raised. When λ is large, the error which remains at

	RKFINT	ADMINT	STFINT	
$\lambda = 0$	301 1×10^{-15} .020	42 7×10^{-16} .015	47(15) 1×10^{-5} .019	NF(NJ) error time
$\lambda = 1$	461 $.4\times10^{-5}$ .038	55 11×10^{-5} .018	49(15) 4×10^{-6} .018	NF(NJ) error time
$\lambda = 10$	1625 $.5\times10^{-5}$ .107	687 9×10^{-5} .135	48(15) 8×10^{-8} .021	NF(NJ) error time
$\lambda = 10^2$	9573 $.5\times10^{-5}$ .58	6402 10×10^{-5} 1.03	52(15) 4×10^{-13} .02	NF(NJ) error time
$\lambda = 10^3$	82745 $.5\times10^{-5}$ 4.86	64849 10×10^{-5} 10.89	54(16) 3×10^{-16} .02	NF(NJ) error time
$\lambda = 10^4$	not attempted	not attempted	51(16) 4×10^{-16} .02	NF(NJ) error time

Table 8-1. Performance of the codes in integrating problem (8-1), illustrating the effect of increasing stiffness. The problem was integrated on the interval [0,50] with output at steps of 1, and error tolerances RELERR=ABSERR=10^{-5}. NF is the number of function evaluations, and NJ the number of Jacobian evaluations, needed for the integration. Error is the maximum relative error actually occurring at any of the output points, and time is the integration time in seconds.

$t = 1$ will be much smaller than 10^{-5}. For RKFINT and ADMINT, errors are not damped out in this fashion because the methods are operating near the boundaries of their stability regions.

Since RKFINT requires that output points be mesh points of the integration, the minimum cost of solving (8-1) with 50 output points is

$$50 \text{ steps} \times 6 \, \frac{\text{function evaluations}}{\text{step}} + 1 \text{ final evaluation}$$

$$= 301 \text{ function evaluations}$$

Thus, the cost of using RKFINT for $\lambda = 0$ and $\lambda = 1$ is largely determined by the number of output points: the integration of these cases would be significantly cheaper if fewer output points were requested.

The second example is the system of three differential equations

$$y_1' = \mu y_1\left(\frac{y_2}{k+y_2}\right) \qquad y_2' = \frac{-\mu}{Z} y_1\left(\frac{y_2}{k+y_2}\right) \qquad y_3' = \left(\frac{1}{Z} - 1\right) \mu y_1\left(\frac{y_2}{k+y_2}\right) \qquad (8\text{-}2)$$

The constants were set to $\mu = 0.04$, $k = 200$, $Z = 0.5$, and the problem was integrated on the interval [0,50], with output at steps of 5, starting with

---

	$\varepsilon = 10^{-4}$	$\varepsilon = 10^{-6}$	$\varepsilon = 10^{-8}$	$\varepsilon = 10^{-10}$	$\varepsilon = 10^{-12}$	
RKFINT	73	91	199	433	1051	NF
	.009	.012	.022	.047	.110	time
ADMINT	38	52	78	104	128	NF
	.016	.020	.027	.036	.046	time
STFINT	60(12)	89(15)	140(20)	209(28)	338(26)	NF(NJ)
	.032	.044	.065	.096	.142	time

Table 8-2. Illustrating the effect of error tolerance on the cost of integrating the problem (8-2), with constants and initial conditions as given in the text. Integration was on the interval [0,50] with output at steps of 5, and error tolerances of RELERR=ABSERR=ε. NF and NJ are the number of function evaluations and the number of Jacobian evaluations, respectively. Integration times are in seconds.

the initial conditions $y_1(0) = 10$, $y_2(0) = 1000$, and $y_3(0) = 0$. This problem is not stiff, and the function evaluations are cheap. Thus, the most efficient integrator is expected to be RKFINT for moderate tolerances and ADMINT for more stringent tolerances, while STFINT should not be competitive. These expectations are verified by the timings shown in Table 8-2.

The final example is the problem

$$\begin{aligned} y_1' &= ay_1 - by_2 + (-a+b+1)e^t \\ y_2' &= by_1 + ay_2 + (-a-b+1)e^t \qquad (8\text{-}3) \\ y_1(0) &= 2 \qquad y_2(0) = 1 \end{aligned}$$

for which the exact solution is

$$\begin{aligned} y_1(t) &= e^{at}\cos bt + e^t \\ y_2(t) &= e^{at}\sin bt + e^t \end{aligned}$$

For this problem, the function evaluations must be regarded as moderately expensive because of the need to evaluate the exponential function e^t. The problem was integrated over the interval [0,10] with output at steps of 0.5 .

Table 8-3 presents the results of the integration for $a = -1$, $b = 2$ using pure relative error tolerances RELERR = ε. This problem is not stiff. Because of the cost of the function evaluations, ADMINT performs more efficiently than RKFINT, except possibly for $\varepsilon = 10^{-4}$. On this machine, the smallest tolerances permitted are $\varepsilon = 10^{-12}$ for RKFINT and 8.89×10^{-16} for ADMINT and STFINT. The errors quoted are the maximum relative errors which occur at any of the output points. Observe that ADMINT is somewhat more successful than STFINT in achieving accuracy near limiting precision.

Table 8-4 shows the results of integrating (8-3) for some different values of a and b. These problems are all stiff to some extent, and STFINT generally performs best. The case $a = -20$ and $b = 70$ provides an excellent example of the fact that the degree of stiffness depends not only on the problem, but also on the amount of accuracy which is sought: for $\varepsilon = 10^{-4}$, STFINT is clearly superior; but at $\varepsilon = 10^{-8}$, ADMINT appears to have a slight advantage.

ε	RKFINT	ADMINT	STFINT	
10^{-4}	247	86	123(12)	NF(NJ)
	$.14\times10^{-4}$	$.67\times10^{-4}$	$.40\times10^{-4}$	error
	.056	.060	.074	time
10^{-6}	493	141	216(15)	NF(NJ)
	$.34\times10^{-6}$	1.8×10^{-6}	$.50\times10^{-6}$	error
	.088	.079	.112	time
10^{-8}	1225	191	404(18)	NF(NJ)
	$.36\times10^{-8}$	1.5×10^{-8}	$.81\times10^{-8}$	error
	.184	.100	.177	time
10^{-10}	3013	316	743(24)	NF(NJ)
	$.41\times10^{-10}$	$.10\times10^{-10}$	1.9×10^{-10}	error
	.38	.13	.29	time
10^{-12}	7423	416	1344(27)	NF(NJ)
	$.43\times10^{-12}$	$.28\times10^{-12}$	4.6×10^{-12}	error
	.92	.15	.50	time
10^{-14}		552	2500(28)	NF(NJ)
		2.7×10^{-14}	8.4×10^{-14}	error
		.20	.89	time
8.89×10^{-16}		972	3674(29)	NF(NJ)
		$24.\times10^{-16}$	184×10^{-16}	error
		.28	1.28	time

Table 8-3. Performance of the codes in integrating problem (8-3) with parameters $a=-1$ and $b=2$. Integration was over the interval [0,10] with output at steps of 0.5, and error tolerances ABSERR=0 and RELERR=ε. NF is the number of function evaluations and NJ is the number of Jacobian evaluations. Integration times are in seconds.

	$a = -20$	$b = 70$		$a = -50$	$b = 50$		
ε	RKFINT	ADMINT	STFINT	RKFINT	ADMINT	STFINT	
10^{-4}	1686	1298	344(28)	1734	1208	223(17)	NF(NJ)
	$.59\times10^{-4}$	2.7×10^{-4}	$.26\times10^{-4}$	$.58\times10^{-4}$	8.3×10^{-4}	$.07\times10^{-4}$	error
	.22	.33	.14	.24	.30	.13	time
10^{-6}	4212	1482	766(37)	4188	1368	420(27)	NF(NJ)
	$.29\times10^{-6}$	5.6×10^{-6}	$.59\times10^{-6}$	$.18\times10^{-6}$	5.8×10^{-6}	$.05\times10^{-6}$	error
	.53	.36	.30	.59	.36	.17	time
10^{-8}	10439	1937	1571(63)	10086	1642	802(50)	NF(NJ)
	$.50\times10^{-8}$	1.3×10^{-8}	1.2×10^{-8}	$.25\times10^{-8}$	1.8×10^{-8}	$.24\times10^{-8}$	error
	1.31	.47	.56	1.25	.41	.30	time
	$a = -100$	$b = 0$		$a = -200$	$b = 100$		
10^{-4}	2093	1384	206(19)	6540	3649	236(24)	NF(NJ)
	$.24\times10^{-4}$	4.4×10^{-4}	$.08\times10^{-4}$	$.17\times10^{-4}$	4.0×10^{-4}	$.02\times10^{-4}$	error
	.30	.37	.10	.77	.97	.10	time
10^{-6}	4901	1540	319(25)	10565	3705	439(30)	NF(NJ)
	$.20\times10^{-6}$	6.9×10^{-6}	$.03\times10^{-6}$	$.34\times10^{-6}$	2.2×10^{-6}	$.03\times10^{-6}$	error
	.64	.40	.16	1.20	.96	.17	time
10^{-8}	11615	1787	599(33)	24863	4069	665(36)	NF(NJ)
	$.43\times10^{-8}$	4.2×10^{-8}	$.05\times10^{-8}$	$.38\times10^{-8}$	4.2×10^{-8}	$.05\times10^{-8}$	error
	1.48	.48	.26	2.94	1.14	.25	time

Table 8-4. Performance of the codes in integrating problem (8-3) with the parameters *a*,*b* as given.

REFERENCES

Ames, W. F. (1969) *Numerical Methods for Partial Differential Equations,* Barnes and Noble, New York

Bettis, D. G. (1978) Embedded Runge-Kutta Methods of Order Four and Five. TICOM Report 78-2, The Texas Institute for Computational Mechanics, University of Texas, Austin

Fehlberg, E. (1969) Low-Order Classical Runge-Kutta Formulas With Stepsize Control and their Application to some Heat Transfer Problems. NASA Technical Report NASA TR R-315

Forsythe, G., and C. Moler (1967) *Computer Solution of Linear Algebraic Systems,* Prentice-Hall, Englewood Cliffs, New Jersey

Gear, C. W. (1971) *Numerical Initial Value Problems in Ordinary Differential Equations,* Prentice-Hall, Englewood Cliffs, New Jersey

Gear, C. W. (1971a) The Automatic Integration of Ordinary Differential Equations. *Communications of the ACM* 14:176-179
Algorithm 407: DIFSUB for Solution of Ordinary Differential Equations. *Communications of the ACM* 14:185-190

Krogh, F. T. (1969) VODQ/SVDQ/DVDQ--Variable Order Integrators for the Numerical Solution of Ordinary Differential Equations. TU Doc. CP-2308, NPO-11643, Jet Propulsion Laboratory, Pasadena

Lassiter, R. R., J. L. Malanchuk, and G. L. Baughman (1976) Comparison of Processes Determining the Fate of Mercury in Aquatic Ecosystems. *Environmental Modeling and Simulation,* U. S. Environmental Protection Agency, Washington. pp. 619-623

Patten, B. C., ed. *Systems Analysis and Simulation in Ecology,* Academic Press, New York
(1971) vol. 1
(1972) vol. 2
(1975) vol. 3
(1976) vol. 4

Schryer, N. L. (1975) A User's Guide to DODES, a Double Precision Ordinary Differential Equation Solver. Bell Laboratories, Murray Hill, New Jersey

Shampine, L. F., and M. K. Gordon (1975) *Computer Solution of Ordinary Differential Equations: The Initial Value Problem,* W. H. Freeman, San Francisco

Shampine, L. F., and H. A. Watts (1976) Practical Solution of Ordinary Differential Equations by Runge-Kutta Methods. Sandia Laboratories Report SAND76-0585, Albuquerque, New Mexico

Shampine, L. F., and K. L. Hiebert (1977) Detecting Stiffness with the Fehlberg (4,5) Formulas. *Comp. and Maths. with Appls.* 3:41-46, Pergamon Press

Appendix A

Runge-Kutta Programs

This appendix contains the complete listings of the subroutines RKFINT and RKFS.

```
      SUBROUTINE RKFINT(NEQN,Y,W,TINIT,TFINAL,TINCR,RELERR,
     1                  ABSERR,IUNIT,FCT,OUTP)
C                                                                C
C****************************************************************C
C                                                                C
C  SUBROUTINE RKFINT USES RKFS TO INTEGRATE A SYSTEM OF NEQN     C
C  FIRST ORDER ORDINARY DIFFERENTIAL EQUATIONS                   C
C                                                                C
C    DY/DT = F(T,Y) ,  Y = (Y(1),Y(2),...,Y(NEQN))               C
C                                                                C
C  FROM TINIT TO TFINAL, WITH OUTPUT OF THE SOLUTION AT          C
C  INTERVALS OF TINCR BY MEANS OF CALLS TO THE USER-             C
C  -SUPPLIED SUBROUTINE OUTP.                                    C
C                                                                C
C                                                                C
C  THE QUANTITIES IN THE ARGUMENT LIST REPRESENT                 C
C                                                                C
C    NEQN -- THE NUMBER OF EQUATIONS                             C
C    Y -- AN ARRAY OF DIMENSION NEQN, WHICH CONTAINS THE         C
C         VALUE OF THE SOLUTION                                  C
C    W -- AN ARRAY OF DIMENSION 7*NEQN, WHICH IS USED AS WORKING C
C         STORAGE BY RKFS.                                       C
C    TINIT -- THE STARTING VALUE OF THE INDEPENDENT VARIABLE     C
C    TFINAL -- THE VALUE OF THE INDEPENDENT VARIABLE AT WHICH    C
C              INTEGRATION IS TO STOP                            C
C    TINCR -- THE DESIRED SPACING, IN THE INDEPENDENT VARIABLE,  C
C             OF THE OUTPUT POINTS                               C
C    RELERR -- THE DESIRED RELATIVE ERROR TOLERANCE              C
C    ABSERR -- THE DESIRED ABSOLUTE ERROR TOLERANCE              C
C    IUNIT -- THE LOGICAL UNIT NUMBER OF THE DATA SET TO BE      C
C             USED FOR WRITING DIAGNOSTIC MESSAGES.  IF IUNIT    C
C             IS LESS THAN OR EQUAL TO ZERO, NO MESSAGES ARE     C
C             WRITTEN                                            C
C    FCT -- THE NAME OF THE USER-SUPPLIED SUBROUTINE             C
C           FCT(T,Y,YP) WHICH EVALUATES THE DERIVATIVES          C
C           YP(I)=DY(I)/DT, I=1,2,...,NEQN                       C
C    OUTP -- THE NAME OF THE USER-SUPPLIED SUBROUTINE            C
```

```
C            OUTP(T,Y,W) USED TO HANDLE OUTPUT.                   C
C                                                                 C
C                                                                 C
C  TO USE RKFINT, THE CALLING PROGRAM MUST                        C
C                                                                 C
C    1. ALLOCATE SPACE FOR THE ARRAYS Y(*) AND W(*) IN A         C
C       DIMENSION STATEMENT.  (NOTE THAT Y MUST BE AN ARRAY,     C
C       EVEN IF NEQN=1.)                                          C
C                                                                 C
C    2. SET Y(1),Y(2),...,Y(NEQN) TO THE INITIAL VALUES TO BE    C
C       USED FOR THE SOLUTION.                                    C
C                                                                 C
C    3. PROVIDE APPROPRIATE VALUES FOR NEQN, TINIT, TFINAL,      C
C       TINCR, RELERR, ABSERR, AND IUNIT.  (SINCE THESE          C
C       QUANTITIES ARE NOT ALTERED BY RKFINT, THEY MAY, IF       C
C       DESIRED, BE WRITTEN AS CONSTANTS IN THE CALL LIST.)      C
C                                                                 C
C    4. PLACE THE ACTUAL NAMES OF FCT AND OUTP IN THE CALL       C
C       LIST, AND DECLARE THOSE NAMES IN AN EXTERNAL STATEMENT.  C
C                                                                 C
C                                                                 C
C  AT EACH CALL TO OUTP(T,Y,W), THE CURRENT VALUES OF THE        C
C  DERIVATIVES DY(I)/DT ARE IN W(1),W(2),...,W(NEQN).  THE       C
C  FIRST CALL TO OUTP IS MADE AT TINIT.                           C
C                                                                 C
C  IT IS NOT NECESSARY THAT THE INTEGRATION INTERVAL BE EVENLY  C
C  DIVISIBLE BY THE OUTPUT MESH SIZE TINCR.  HOWEVER, IF IT IS  C
C  NOT, INTEGRATION WILL PROCEED TO THE FIRST MESH POINT          C
C  BEYOND TFINAL, RATHER THAN STOPPING AT TFINAL.                 C
C                                                                 C
C  UNDER CERTAIN CIRCUMSTANCES, THE OUTPUT ROUTINE OUTP MAY      C
C  BE USED TO ALTER SOME OF THE PARAMETERS IN THE COMMON          C
C  BLOCK RKDATA.  SEE THE PROGRAM DOCUMENTATION FOR A             C
C  DISCUSSION OF THIS.                                            C
C                                                                 C
C*****************************************************************C
C                                                                 C
C  THIS SUBROUTINE IS WRITTEN IN SINGLE PRECISION.                C
C  A DOUBLE PRECISION VERSION FOR USE WITH A DOUBLE PRECISION    C
C  VERSION OF RKFS MAY BE OBTAINED SIMPLY BY REMOVING THE C       C
C  FROM COLUMN 1 OF THE NEXT FIVE CARDS.                          C
C     DOUBLE PRECISION ABS,ABSER,ABSERR,DABS,DSIGN,DT,D1,D2,
C    1                 EPSDT,H,RELER,RELERR,SIGN,T,TFINAL,TINCR,
C    2                 TINIT,TMAXIN,TOUT,W,Y
C     ABS(D1) = DABS(D1)
C     SIGN(D1,D2) = DSIGN(D1,D2)
C                                                                 C
      COMMON /RKDATA/ T,TOUT,RELER,ABSER,H,NEQ,IFLAG,NFE,KOP,
     1                 INIT,NSS,KST
C                                                                 C
      DIMENSION Y(NEQN),W(1)
C                                                                 C
```

```
C  THE ACTUAL DIMENSION OF W, AS DEFINED BY THE CALLING PROGRAM,       C
C  MUST BE AT LEAST 7*NEQN.                                            C
C                                                                      C
C                                                                      C
      EXTERNAL FCT
C                                                                      C
C  WRITE STARTING DATA INTO MESSAGE FILE                               C
C                                                                      C
      IF (IUNIT.GT.0) WRITE(IUNIT,100) NEQN,TINIT,TFINAL,TINCR,
     1                                        RELERR,ABSERR
C                                                                      C
C  COMPUTE INDICES FOR SPLITTING THE WORK ARRAY W(*).                  C
C                                                                      C
      K1=NEQN+1
      K2=K1+NEQN
      K3=K2+NEQN
      K4=K3+NEQN
      K5=K4+NEQN
      K6=K5+NEQN
C                                                                      C
C  INITIALIZE VARIABLES TO START INTEGRATION WITH RKFS.                C
C                                                                      C
      NEQ=NEQN
      RELER=RELERR
      ABSER=ABSERR
      IFLAG=1
      T=TINIT
      TOUT=TINIT
C                                                                      C
C  SAVE INFORMATION NEEDED TO LOCATE END OF INTEGRATION.               C
C                                                                      C
      TMAXIN=ABS(TFINAL-TINIT)
      EPSDT=0.001*ABS(TINCR)
C                                                                      C
C  SET SIGN OF OUTPUT INTERVAL TO PROCEED FROM TINIT TO TFINAL.        C
C                                                                      C
      DT=SIGN(TINCR,TFINAL-TINIT)
C                                                                      C
C  INITIALIZE COUNTER OF ERROR RETURNS.                                C
C                                                                      C
      IER=0
C                                                                      C
C**********************************************************************C
C                                                                      C
C  CALL RKFS TO INTEGRATE TO THE NEXT OUTPUT POINT.                    C
C                                                                      C
    5 CALL RKFS(FCT,Y,W(1),W(K1),W(K2),W(K3),W(K4),W(K5),W(K6))
      IF (IFLAG.NE.2) GO TO 8
C                                                                      C
C  SUCCESSFUL INTEGRATION TO OUTPUT POINT.                             C
C                                                                      C
      CALL OUTP(T,Y,W)
```

```
C                                                                C
C  CHECK WHETHER TFINAL HAS BEEN REACHED OR PASSED.              C
C                                                                C
      IF (ABS(T-TINIT).GE.TMAXIN) GO TO 95
C                                                                C
C  INTEGRATION IS NOT YET COMPLETE, SO THE NEXT OUTPUT POINT IS  C
C  SET.  IF THE NEW POINT DIFFERS FROM TFINAL BY AN AMOUNT       C
C  ATTRIBUTABLE TO ROUNDOFF ERROR, IT IS RESET TO TFINAL EXACTLY.C
C  INTEGRATION IS THEN CONTINUED.                                C
C                                                                C
      TOUT=T+DT
      IF (ABS(TFINAL-TOUT).LT.EPSDT) TOUT=TFINAL
      GO TO 5
C                                                                C
C****************************************************************C
C                                                                C
C  INTEGRATION WAS INTERRUPTED BEFORE REACHING THE OUTPUT POINT. C
C  INCREMENT THE ERROR COUNTER AND BRANCH FOR WRITING THE        C
C  APPROPRIATE DIAGNOSTIC MESSAGE.  INTEGRATION WILL THEN BE     C
C  CONTINUED IF THE ERROR IS NOT TOO SEVERE.                     C
C                                                                C
    8 IER=IER+1
      GO TO (20,20,30,40,50,60,70,80,90),IFLAG
C                                                                C
C                                                                C
C  IFLAG HAS AN ILLEGAL VALUE, SO THAT A PRECISE DEFINITION OF   C
C  THE ERROR IS NOT POSSIBLE, AND EXECUTION MUST BE TERMINATED.  C
C                                                                C
   20 IF (IUNIT.GT.0) WRITE(IUNIT,200) T,IFLAG
      STOP
C                                                                C
C                                                                C
C  IFLAG = 3 -- THE REQUESTED RELATIVE ERROR TOLERANCE WAS TOO   C
C  SMALL FOR THE MACHINE PRECISION AND HAS BEEN INCREASED TO     C
C  THE MINIMUM PERMISSIBLE VALUE.                                C
C                                                                C
   30 IF (IUNIT.GT.0) WRITE(IUNIT,300) T,RELER
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C  IFLAG = 4 -- THE REQUESTED ERROR TOLERANCES WERE TOO          C
C  STRINGENT FOR THIS PROBLEM AND HAVE BEEN INCREASED TO         C
C  VALUES WHICH SHOULD BE ACCEPTABLE.                            C
C                                                                C
   40 IF (IUNIT.GT.0) WRITE(IUNIT,400) T,RELER,ABSER
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 5 -- THE COUNTER OF CALLS TO FCT WAS RESET AFTER      C
C  EXCEEDING THE LIMIT IMPOSED IN RKFS.                          C
C                                                                C
```

```
   50 IF (IUNIT.GT.0) WRITE(IUNIT,500) T
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 6 -- THE COUNTER OF CALLS TO FCT WAS RESET AFTER      C
C  STIFFNESS CAUSED IT TO EXCEED THE LIMIT IMPOSED IN RKFS.      C
C                                                                C
   60 IF (IUNIT.GT.0) WRITE(IUNIT,600) T
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 7 -- THE MONITOR OF OUTPUT FREQUENCY WAS RESET        C
C  AFTER INDICATING THAT THE OUTPUT INTERVAL IS TOO SMALL        C
C  TO ALLOW EFFICIENT OPERATION OF RKFS.                         C
C                                                                C
   70 IF (IUNIT.GT.0) WRITE(IUNIT,700) T
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 8 -- FATAL ERROR -- A COMPONENT OF THE SOLUTION       C
C  VANISHED, MAKING A PURE RELATIVE ERROR TEST IMPOSSIBLE.       C
C  THE USER MUST SUPPLY AN APPROPRIATE NONZERO VALUE OF          C
C  ABSERR FOR SUCCESSFUL INTEGRATION.                            C
C                                                                C
   80 IF (IUNIT.GT.0) WRITE(IUNIT,800) T
      STOP
C                                                                C
C                                                                C
C  IFLAG = 9 -- FATAL ERROR -- AN ILLEGAL PARAMETER VALUE        C
C  WAS PASSED TO RKFS.  THE USER MUST LOCATE AND CORRECT THE     C
C  ERROR.                                                        C
C                                                                C
   90 IF (IUNIT.GT.0) WRITE(IUNIT,900) T,NEQ,RELER,ABSER,TINCR
      STOP
C                                                                C
C                                                                C
C  THE PROGRAM IS STOPPED BECAUSE THERE HAVE BEEN FIVE ERROR     C
C  RETURNS FROM RKFS, INDICATING SERIOUS PROBLEMS WHICH          C
C  REQUIRE INTERVENTION BY THE USER.                             C
C                                                                C
   92 IF (IUNIT.GT.0) WRITE(IUNIT,920) T
      STOP
C                                                                C
C                                                                C
C  THE INTEGRATION HAS BEEN COMPLETED TO OR BEYOND TFINAL.       C
C                                                                C
   95 IF (IUNIT.GT.0) WRITE(IUNIT,950) T
      RETURN
C                                                                C
```

```
C**********************************************************************C
C                                                                      C
C  THE REMAINDER OF THE PROGRAM CONTAINS THE FORMAT STATEMENTS         C
C  FOR THE DIAGNOSTIC MESSAGES.                                        C
C                                                                      C
  100 FORMAT(36H RKFINT HAS BEEN CALLED TO INTEGRATE,I5,
     1 10H EQUATIONS/18H FROM INITIAL TIME,G12.5/
     2 14H TO FINAL TIME,G12.5/
     3 24H WITH OUTPUT AT STEPS OF,G12.5/
     4 32H AND INITIAL ERROR TOLERANCES OF/
     5 9H RELERR =,G12.5,9H ABSERR =,G12.5/)
C                                                                      C
  200 FORMAT(4H ***,12(4H****)/
     1 37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2 52H *** BECAUSE AN UNIDENTIFIABLE ERROR HAS CAUSED  ***/
     3 52H *** RKFS TO RETURN WITH THE ILLEGAL FLAG VALUE  ***/
     4 12H *** IFLAG =,I11,26X,3H***/4H ***,12(4H****))
C                                                                      C
  300 FORMAT(17H *WARNING--AT T =,G12.5,21H, THE VALUE OF RELERR/
     1 49H PASSED TO RKFS WAS FOUND TO BE TOO SMALL FOR THE/
     2 45H MACHINE PRECISION.  IT HAS BEEN INCREASED TO/
     3 9H RELERR =,G12.5,23H TO CONTINUE EXECUTION./)
C                                                                      C
  400 FORMAT(17H *WARNING--AT T =,G12.5,20H, RKFS WAS UNABLE TO/
     1 52H MEET THE REQUESTED ERROR TOLERANCES AT THE SMALLEST/
     2 52H ALLOWABLE STEPSIZE.  INTEGRATION IS CONTINUING WITH/
     3 37H THE TOLERANCES INCREASED TO RELERR =,G12.5/
     4 13H AND ABSERR =,G12.5/)
C                                                                      C
  500 FORMAT(17H *WARNING--AT T =,G12.5,24H, THE NUMBER OF FUNCTION/
     1 51H CALLS EXCEEDED THE MAXIMUM NUMBER, MAXNFE, ALLOWED/
     2 51H BY RKFS.  THE FUNCTION EVALUATION COUNTER HAS BEEN/
     3 51H REDUCED BY MAXNFE TO ALLOW INTEGRATION TO PROCEED./)
C                                                                      C
  600 FORMAT(17H *WARNING--AT T =,G12.5,22H, STIFFNESS HAS CAUSED/
     1 51H THE NUMBER OF FUNCTION CALLS TO EXCEED THE MAXIMUM/
     2 47H NUMBER, MAXNFE, ALLOWED BY RKFS.  THE FUNCTION/
     3 49H EVALUATION COUNTER HAS BEEN REDUCED BY MAXNFE TO/
     4 30H ALLOW INTEGRATION TO PROCEED./)
C                                                                      C
  700 FORMAT(17H *WARNING--AT T =,G12.5,23H, RKFS RETURNED WITH AN/
     1 50H INDICATION THAT THIS INTEGRATION IS UNNECESSARILY/
     2 48H COSTLY BECAUSE THE REQUESTED OUTPUT INTERVAL IS/
     3 49H SUBSTANTIALLY SMALLER THAN THE NATURAL STEPSIZE./
     4 48H (REFER TO PROGRAM DOCUMENTATION FOR ALTERNATIVE/
     5 52H APPROACHES.)  THE OUTPUT FREQUENCY MONITOR HAS BEEN/
     6 39H RESET TO ALLOW INTEGRATION TO PROCEED./)
C                                                                      C
  800 FORMAT(4H ***,12(4H****)/
     1 37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2 40H *** BECAUSE A COMPONENT OF THE SOLUTION,9X,3H***/
     3 52H *** VANISHED WHILE ABSERR = 0.  IT WILL BE      ***/
     4  52H *** NECESSARY TO PROVIDE A NONZERO ABSOLUTE     ***/
     5  52H *** ERROR TOLERANCE TO PERFORM THIS INTEGRATION.***/
     6  4H ***,12(4H****))
C                                                                      C
  900 FORMAT(4H ***,12(4H****)/
     1  37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2  52H *** BECAUSE INVALID INPUT WAS RECEIVED BY RKFS. ***/
     3  37H *** CHECK THE FOLLOWING PARAMETERS--,12X,3H***/
     4  12H ***  NEQN =,I11,26X,3H***/
     5  14H ***  RELERR =,G12.5,23X,3H***/
     6  14H ***  ABSERR =,G12.5,23X,3H***/
     7  13H ***  TINCR =,G12.5,24X,3H***/
     8  52H *** ALSO, CHECK THAT THE OUTPUT ROUTINE DOES    ***/
     9  41H *** NOT RESET IFLAG TO AN ILLEGAL VALUE.,8X,3H***/
     A  4H ***,12(4H****))
C                                                                      C
  920 FORMAT(4H ***,12(4H****)/
     1  37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2  52H *** BECAUSE RKFS HAS RETURNED IRREGULARLY FIVE  ***/
     3  52H *** TIMES, WHICH INDICATES SOME SORT OF EXTREME ***/
     4  52H *** DIFFICULTY IN THE INTEGRATION.  THE USER    ***/
     5  52H *** SHOULD CAREFULLY CHECK THE ACCURACY OF HIS  ***/
     6  52H *** PROBLEM STATEMENT AND CODING.  IT IS ALSO   ***/
     7  52H *** POSSIBLE THAT THIS PROBLEM IS TOO STIFF TO  ***/
     8  27H *** BE INTEGRATED BY RKFS.,22X,3H***/
     9  4H ***,12(4H****))
C                                                                      C
  950 FORMAT(4H ***,12(4H****)/
     1  32H *** INTEGRATION COMPLETE AT T =,G12.5,5X,3H***/
     2  4H ***,12(4H****)///)
C                                                                      C
C**********************************************************************C
C                                                                      C
C  END OF RKFINT                                                       C
C                                                                      C
      END


      SUBROUTINE RKFS(F,Y,YP,F1,F2,F3,F4,F5,W)
C                                                                      C
C**********************************************************************C
C                                                                      C
C  SUBROUTINE RKFS USES THE FEHLBERG FOURTH-FIFTH ORDER RUNGE-         C
C  -KUTTA METHOD TO INTEGRATE A SYSTEM OF NEQN FIRST ORDER             C
C  ORDINARY DIFFERENTIAL EQUATIONS                                     C
C                                                                      C
C    DY/DT = F(T,Y) ,  Y = (Y(1),Y(2),...,Y(NEQN))                     C
C                                                                      C
C  FROM T TO TOUT.  RKFS IS NORMALLY USED IN 'INTERVAL MODE', IN       C
C  WHICH IT RETURNS WITH THE SOLUTION EVALUATED AT TOUT.  IT MAY       C
C  ALSO BE USED IN 'SINGLE-STEP MODE', IN WHICH IT RETURNS WITH        C
C  THE SOLUTION ADVANCED ONE STEP IN THE DIRECTION OF TOUT.            C
```

```
C                                                                C
C  THE PARAMETERS IN THE ARGUMENT LIST ARE                       C
C                                                                C
C    F -- THE NAME OF THE USER-SUPPLIED SUBROUTINE F(T,Y,YP),    C
C         WHICH EVALUATES THE DERIVATIVES YP(I)=DY(I)/DT,        C
C         I=1,2,...,NEQN.                                        C
C    Y -- AN ARRAY OF DIMENSION NEQN, WHICH CONTAINS THE VALUE   C
C         OF THE SOLUTION.                                       C
C    YP -- AN ARRAY OF DIMENSION NEQN, WHICH CONTAINS THE VALUE  C
C          OF THE DERIVATIVE DY/DT.                              C
C    F1,F2,F3,F4,F5,W -- ARRAYS EACH OF DIMENSION NEQN, WHICH    C
C                        ARE USED FOR WORKING STORAGE BY RKFS.   C
C                                                                C
C  THE PARAMETERS IN THE COMMON BLOCK RKDATA ARE                 C
C                                                                C
C    T -- THE INDEPENDENT VARIABLE                               C
C                                                                C
C    TOUT -- THE VALUE OF T AT WHICH OUTPUT IS DESIRED           C
C                                                                C
C    RELERR,ABSERR -- RELATIVE AND ABSOLUTE ERROR TOLERANCES FOR C
C       THE LOCAL ERROR TEST.  EACH STEPSIZE IS CHOSEN SO THAT   C
C       ABS(LOCAL ERROR(I)) .LE. RELERR*ABS(Y(I))+ABSERR FOR     C
C       EACH COMPONENT OF THE SOLUTION.                          C
C    NEQN -- THE NUMBER OF EQUATIONS                             C
C                                                                C
C    IFLAG -- INDICATOR OF THE STATUS OF INTEGRATION             C
C                                                                C
C    H,NFE,KOP,INIT,NSS,KST -- THOSE QUANTITIES USED FOR         C
C       INTERNAL BOOKKEEPING BY RKFS WHICH ARE NEEDED FOR        C
C       CONTINUING INTEGRATION, BUT WHICH ARE NORMALLY OF NO     C
C       INTEREST TO THE USER.  THEY ARE PLACED IN THE COMMON     C
C       BLOCK TO AVOID LOCAL RETENTION BY RKFS BETWEEN CALLS.   C
C       SEE THE PROGRAM DOCUMENTATION FOR A DESCRIPTION OF THEIR C
C       FUNCTIONING.                                             C
C                                                                C
C----------------------------------------------------------------C
C                                                                C
C  TO USE RKFS, THE CALLING PROGRAM MUST                         C
C                                                                C
C    1. ALLOCATE SPACE FOR THE ARRAYS Y(*), YP(*), F1(*), F2(*), C
C       F3(*), F4(*), F5(*), W(*) IN A DIMENSION STATEMENT.      C
C       (THESE MUST BE ARRAYS EVEN IF NEQN=1.)                   C
C                                                                C
C    2. DECLARE THE NAME OF F IN AN EXTERNAL STATEMENT, AND      C
C       PLACE THAT NAME IN THE CALL LIST.                        C
C                                                                C
C                                                                C
C  FIRST CALL TO RKFS                                            C
C  ------------------                                            C
C                                                                C
C  TO BEGIN INTEGRATION, THE CALLING PROGRAM MUST                C
C                                                                C
```

```
C    1. INITIALIZE THE FOLLOWING VARIABLES IN THE COMMON BLOCK   C       16600
C       RKDATA                                                   C       16700
C         T -- THE STARTING VALUE OF THE INDEPENDENT VARIABLE    C       16800
C         TOUT -- THE VALUE OF THE INDEPENDENT VARIABLE AT WHICH C       16900
C            OUTPUT IS DESIRED.  (TOUT=T IS PERMITTED ON THE     C       17000
C            FIRST CALL ONLY, IN WHICH CASE RKFS EVALUATES THE   C       17100
C            DERIVATIVES YP AND RETURNS WITH IFLAG=2.)           C       17200
C         RELERR -- THE DESIRED RELATIVE ERROR TOLERANCE         C       17300
C         ABSERR -- THE DESIRED ABSOLUTE ERROR TOLERANCE         C       17400
C         IFLAG -- +1 FOR INTERVAL MODE OR -1 FOR SINGLE-STEP    C       17500
C            MODE.                                               C       17600
C                                                                C       17700
C    2. INITIALIZE Y(*) TO CONTAIN THE STARTING VALUE OF THE     C       17800
C       SOLUTION.                                                C       17900
C                                                                C       18000
C                                                                C       18100
C  UPON RETURN, T IS THE LAST VALUE OF THE INDEPENDENT VARIABLE  C       18200
C  SUCCESSFULLY REACHED DURING THE INTEGRATION, Y CONTAINS THE   C       18300
C  SOLUTION AT T, YP CONTAINS THE DERIVATIVES AT T, AND IFLAG    C       18400
C  INDICATES THE STATUS OF THE INTEGRATION AS DESCRIBED BELOW.   C       18500
C                                                                C       18600
C                                                                C       18700
C  SUBSEQUENT CALLS TO RKFS                                      C       18800
C  ------------------------                                      C       18900
C                                                                C       19000
C  RKFS NORMALLY RETURNS WITH ALL INFORMATION NEEDED TO CONTINUE C       19100
C  INTEGRATION.                                                  C       19200
C                                                                C       19300
C  INTERVAL MODE --                                              C       19400
C                                                                C       19500
C    IF THE INTEGRATION REACHED TOUT (IFLAG=2), THE USER NEED    C       19600
C    ONLY DEFINE A NEW TOUT AND CALL RKFS AGAIN.                 C       19700
C                                                                C       19800
C    IF THE INTEGRATION WAS INTERRUPTED WITH IFLAG = 3, 4, 5, 6, C       19900
C    OR 7, INTEGRATION MAY BE CONTINUED SIMPLY BY CALLING RKFS   C       20000
C    AGAIN.  HOWEVER, SUCH A RETURN INDICATES THAT THE           C       20100
C    INTEGRATION IS NOT PROCEEDING SO SMOOTHLY AS MIGHT BE       C       20200
C    EXPECTED, SO THE USER SHOULD EXERCISE SOME CAUTION.         C       20300
C                                                                C       20400
C    IF THE INTEGRATION WAS INTERRUPTED WITH IFLAG = 8, THE USER C       20500
C    MUST GIVE ABSERR A NONZERO POSITIVE VALUE AND SET IFLAG = 2 C       20600
C    IN ORDER TO CONTINUE.                                       C       20700
C                                                                C       20800
C    IF RKFS RETURNED WITH IFLAG = 9, THE USER MUST CORRECT THE  C       20900
C    INVALID INPUT AND SET IFLAG = 2 (OR IFLAG = 1, IF           C       21000
C    INTEGRATION HAS NOT YET STARTED) IN ORDER TO CONTINUE.      C       21100
C                                                                C       21200
C    THE USER MAY AT ANY TIME SWITCH TO SINGLE-STEP MODE BY      C       21300
C    REVERSING THE SIGN OF IFLAG.                                C       21400
C                                                                C       21500
C  SINGLE-STEP MODE --                                           C       21600
C                                                                C       21700
```

```
C    IF THE INTEGRATION HAS PROCEEDED ONE STEP (IFLAG=-2), THE    C
C    USER NEED ONLY CALL RKFS AGAIN TO PROCEED ANOTHER STEP.      C
C                                                                 C
C    IF THE INTEGRATION HAS REACHED TOUT (IFLAG=2), THE USER      C
C    MUST DEFINE A NEW TOUT AND SET IFLAG = -2 IN ORDER TO        C
C    CONTINUE IN SINGLE-STEP MODE.  IF IFLAG IS NOT RESET,        C
C    INTEGRATION WILL PROCEED IN INTERVAL MODE.                   C
C                                                                 C
C    IF THE INTEGRATION WAS INTERRUPTED WITH IFLAG = -3, -4, -5,  C
C    -6, OR -7, INTEGRATION MAY BE CONTINUED SIMPLY BY CALLING    C
C    RKFS AGAIN.                                                  C
C                                                                 C
C    IF THE INTEGRATION WAS INTERRUPTED WITH IFLAG = 8, THE USER  C
C    MUST GIVE ABSERR A NONZERO POSITIVE VALUE AND SET IFLAG =-2  C
C    IN ORDER TO CONTINUE.                                        C
C                                                                 C
C    IF RKFS RETURNED WITH IFLAG = 9, THE USER MUST CORRECT THE   C
C    INVALID INPUT AND SET IFLAG =-2 (OR IFLAG =-1, IF            C
C    INTEGRATION HAS NOT YET STARTED) IN ORDER TO CONTINUE.       C
C                                                                 C
C    THE USER MAY AT ANY TIME SWITCH TO INTERVAL MODE BY          C
C    REVERSING THE SIGN OF A NEGATIVE IFLAG.                      C
C                                                                 C
C-----------------------------------------------------------------C
C  UPON RETURN, THE VALUE OF IFLAG INDICATES THE STATUS OF THE    C
C  INTEGRATION AS FOLLOWS,                                        C
C                                                                 C
C  IFLAG = 2 -- T=TOUT, SO THAT INTEGRATION WAS SUCCESSFUL TO     C
C               THE OUTPUT POINT.                                 C
C  IFLAG = 3 -- INTEGRATION WAS NOT ATTEMPTED BECAUSE THE         C
C               REQUESTED ERROR TOLERANCE RELERR WAS SMALLER      C
C               THAN THE MINIMUM PERMISSIBLE VALUE REMIN.         C
C               RELERR HAS BEEN RESET TO REMIN.                   C
C  IFLAG = 4 -- INTEGRATION WAS INTERRUPTED BECAUSE THE           C
C               REQUESTED ERROR TOLERANCES COULD NOT BE           C
C               ATTAINED AT THE SMALLEST ALLOWABLE STEPSIZE.      C
C               THE TOLERANCES HAVE BEEN INCREASED TO VALUES      C
C               WHICH SHOULD ALLOW SUCCESSFUL CONTINUATION.       C
C  IFLAG = 5 -- INTEGRATION WAS INTERRUPTED AFTER THE COUNTER     C
C               OF CALLS TO F EXCEEDED MAXNFE.  THE COUNTER HAS   C
C               BEEN REDUCED BY MAXNFE.                           C
C  IFLAG = 6 -- SAME AS IFLAG=5, EXCEPT THAT THE PROBLEM HAS      C
C               BEEN ADDITIONALLY DIAGNOSED AS STIFF.             C
C  IFLAG = 7 -- INTEGRATION WAS NOT CONTINUED BECAUSE THE         C
C               DISTANCE FROM T TO TOUT HAS BEEN LESS THAN HALF   C
C               THE NATURAL STEPSIZE FOR 100 CALLS TO RKFS,       C
C               INDICATING THAT THE CODE IS BEING USED IN A WAY   C
C               WHICH GREATLY IMPAIRS ITS EFFICIENCY.  THE        C
C               MONITOR OF THIS EFFECT WAS RESET TO ALLOW         C
C               CONTINUATION IF DESIRED.                          C
C  IFLAG = 8 -- INTEGRATION WAS INTERRUPTED WHEN A COMPONENT OF   C
C               THE SOLUTION VANISHED WHILE A PURE RELATIVE       C
```

```
C               ERROR TEST WAS SPECIFIED.  THE USER MUST SUPPLY  C
C               A NONZERO ABSERR TO CONTINUE INTEGRATION.        C
C  IFLAG = 9 -- INVALID INPUT WAS PASSED TO RKFS.  THE USER MUST C
C               LOCATE AND CORRECT THE ERROR IN ORDER TO         C
C               CONTINUE INTEGRATION.                            C
C  IFLAG =-2 -- A SINGLE SUCCESSFUL STEP HAS BEEN TAKEN IN THE   C
C               DIRECTION OF TOUT, BUT TOUT HAS NOT YET BEEN     C
C               REACHED.  THIS OCCURS ONLY IF RKFS WAS CALLED    C
C               WITH IFLAG NEGATIVE.                             C
C  IFLAG =-3, -4, -5, -6, OR -7 -- SAME AS IFLAG = 3, 4, 5, 6,   C
C               OR 7, RESPECTIVELY, EXCEPT THAT RKFS WAS CALLED  C
C               WITH IFLAG NEGATIVE AND IS OPERATING IN SINGLE-  C
C               -STEP MODE.                                      C
C                                                                C
C****************************************************************C
C                                                                C
C  THIS SUBROUTINE IS WRITTEN IN SINGLE PRECISION.               C
C  WITH MOST COMPILERS, A DOUBLE PRECISION VERSION MAY BE        C
C  OBTAINED SIMPLY BY SETTING APPROPRIATE VALUES FOR U26         C
C  AND REMIN IN THE DATA STATEMENT BELOW AND BY REMOVING         C
C  THE C FROM COLUMN 1 OF THE NEXT EIGHT CARDS.                  C
C     DOUBLE PRECISION ABS,ABSERR,AE,AMAX1,CH,DABS,DMAX1,DSIGN,
C    1                 DT,D1,D2,EE,EEC,EECOET,EEOET,ESTTOL,ET,
C    2                 F1,F2,F3,F4,F5,GFACT,H,HMIN,RELERR,REMIN,
C    3                 S,SCALE,SIGN,T,TOL,TOLN,TOUT,U26,
C    4                 Y,YP,YPK,W
C     ABS(D1) = DABS(D1)
C     SIGN(D1,D2) = DSIGN(D1,D2)
C     AMAX1(D1,D2) = DMAX1(D1,D2)
C                                                                C
      LOGICAL HFAILD,OUTPUT,STIFF,NOTEST
C                                                                C
      DIMENSION Y(NEQN),YP(NEQN),F1(NEQN),F2(NEQN),F3(NEQN),
     1          F4(NEQN),F5(NEQN),W(NEQN)
C                                                                C
      COMMON /RKDATA/ T,TOUT,RELERR,ABSERR,H,NEQN,IFLAG,NFE,KOP,
     1                INIT,NSS,KST
C                                                                C
C                                                                C
C  IN THE FOLLOWING DATA STATEMENT, U26 AND REMIN MUST BE SET    C
C  TO VALUES APPROPRIATE TO THE MACHINE BEING USED.              C
C                                                                C
C  U26 IS 26.*U, WHERE THE COMPUTER UNIT ROUNDOFF ERROR U IS     C
C  THE SMALLEST POSITIVE VALUE REPRESENTABLE IN THE MACHINE      C
C  SUCH THAT (1.+U).GT.1.                                        C
C                                                                C
C  REMIN IS A TOLERANCE THRESHOLD WHICH SHOULD NOT BE SMALLER    C
C  THAN 4.*U.  SINCE THIS INTEGRATION METHOD CANNOT OPERATE      C
C  EFFICIENTLY AT EXTREME TOLERANCES, REMIN SHOULD BE CHOSEN     C
C  AS THE LARGER OF 4.*U AND 1.E-12                              C
C                                                                C
C  SOME REPRESENTATIVE VALUES ARE                                C
```

```
C    U=9.54E-7   U26=2.48E-5   REMIN=3.82E-6  IBM 360/370 S.P.   C
C    U=5.96E-8   U26=1.55E-6   REMIN=2.39E-7  PDP-11 S.P.        C
C    U=1.49E-8   U26=3.88E-7   REMIN=5.97E-8  UNIVAC 1108        C
C    U=7.45E-9   U26=1.94E-7   REMIN=2.99E-8  PDP-10             C
C    U=7.11E-15  U26=1.85E-13  REMIN=1.E-12   CDC 6000 SERIES    C
C    U=2.22D-16  U26=5.78D-15  REMIN=1.D-12   IBM 360/370 D.P.   C
C    U=1.39D-17  U26=3.61D-16  REMIN=1.D-12   PDP-11 D.P.        C
C                                                                C
      DATA U26/1.55E-6/,REMIN/2.39E-7/
C                                                                C
C                                                                C
C  THE EXPENSE IS CONTROLLED BY RESTRICTING THE NUMBER           C
C  OF FUNCTION EVALUATIONS TO BE APPROXIMATELY MAXNFE.           C
C  AS SET, THIS CORRESPONDS TO ABOUT 1000 STEPS.                 C
C                                                                C
      DATA MAXNFE/6000/
C                                                                C
C****************************************************************C
C                                                                C
C  CHECK INPUT PARAMETERS                                        C
C                                                                C
      MFLAG=IABS(IFLAG)
      DT=TOUT-T
C                                                                C
C  THE INPUT IS INVALID IF NEQN IS LESS THAN 1, IF IFLAG IS ZERO C
C  OR HAS ABSOLUTE VALUE GREATER THAN 7, OR IF EITHER OF RELERR  C
C  AND ABSERR IS NEGATIVE.                                       C
C                                                                C
      IF ((MFLAG.GT.7).OR.(MFLAG.EQ.0).OR.(NEQN.LT.1)) GO TO 430
      IF ((RELERR.LT.0.0).OR.(ABSERR.LT.0.0)) GO TO 430
C                                                                C
C  IF THIS IS THE FIRST CALL, GO TO INITIALIZATION SECTION.      C
C                                                                C
      IF (MFLAG.EQ.1) GO TO 400
C                                                                C
C  SINCE THIS IS NOT THE FIRST CALL, T = TOUT IS NOT VALID INPUT.C
C                                                                C
      IF (T.EQ.TOUT) GO TO 430
C                                                                C
C  IF NO STEP HAS YET BEEN ATTEMPTED, GO TO INITIALIZATION       C
C  SECTION TO COMPUTE THE STARTING STEPSIZE.                     C
C                                                                C
      IF (INIT.EQ.0) GO TO 410
C                                                                C
C  BEFORE CONTINUING INTEGRATION, CHECK WHETHER REQUESTED ERROR  C
C  TOLERANCE IS TOO SMALL FOR THE MACHINE PRECISION.             C
C                                                                C
      IF (RELERR.LT.REMIN) GO TO 440
C                                                                C
C  SET STEPSIZE FOR INTEGRATION IN THE DIRECTION FROM T TO TOUT  C
C                                                                C
   80 H=SIGN(H,DT)
```

```
C                                                                C
C  TEST WHETHER THE EFFICIENCY OF INTEGRATION IS BEING SEVERELY  C
C  IMPAIRED BY TOO FREQUENT OUTPUT POINTS.                       C
C                                                                C
      IF (ABS(H).GE.2.0*ABS(DT)) KOP=KOP+1
      IF (KOP.GT.100) GO TO 450
C                                                                C
C  TEST WHETHER THE INTERVAL FROM T TO TOUT IS TOO SMALL FOR     C
C  MINIMUM ALLOWABLE STEPSIZE.                                   C
C                                                                C
      IF (ABS(DT).LT.U26*ABS(T)) GO TO 460
C                                                                C
C  SET INDICATORS FOR STIFFNESS TEST                             C
C                                                                C
      STIFF=.FALSE.
      IF (KST.GE.25) STIFF=.TRUE.
      NOTEST=STIFF
      IF ((NSS-KST).GT.25) NOTEST=.TRUE.
C                                                                C
C  INITIALIZE OUTPUT POINT INDICATOR                             C
C                                                                C
      OUTPUT=.FALSE.
C                                                                C
C  TO AVOID PREMATURE UNDERFLOW IN THE ERROR TOLERANCE FUNCTION, C
C  SCALE THE ERROR TOLERANCES                                    C
C                                                                C
      SCALE=2.0/RELERR
      AE=SCALE*ABSERR
C                                                                C
C****************************************************************C
C****************************************************************C
C                                                                C
C  STEP BY STEP INTEGRATION                                      C
C                                                                C
  100 HFAILD=.FALSE.
C                                                                C
C  SET SMALLEST ALLOWABLE STEPSIZE                               C
C                                                                C
      HMIN=U26*ABS(T)
C                                                                C
C  ADJUST STEPSIZE IF NECESSARY TO HIT THE OUTPUT POINT.         C
C  LOOK AHEAD TWO STEPS TO AVOID DRASTIC CHANGES IN THE STEPSIZE C
C  AND THUS LESSEN THE IMPACT OF OUTPUT POINTS ON THE CODE.      C
C                                                                C
      DT=TOUT-T
      IF (ABS(DT).GE.2.0*ABS(H)) GO TO 200
      IF (ABS(DT).GT.ABS(H)/0.9) GO TO 150
C                                                                C
C  THE NEXT SUCCESSFUL STEP WILL COMPLETE THE INTEGRATION TO     C
C  THE OUTPUT POINT                                              C
C                                                                C
      OUTPUT=.TRUE.
      H=DT
      GO TO 200
C                                                                 C
  150 H=0.5*DT
C                                                                 C
C*****************************************************************C
C  CORE INTEGRATOR FOR TAKING A SINGLE STEP                       C
C*****************************************************************C
C                                                                 C
C  THE TOLERANCES HAVE BEEN SCALED TO AVOID PREMATURE UNDERFLOW   C
C  IN COMPUTING THE ERROR TOLERANCE FUNCTION ET.                  C
C                                                                 C
C  TO AVOID PROBLEMS WITH ZERO CROSSINGS, RELATIVE ERROR IS       C
C  MEASURED USING THE AVERAGE OF THE MAGNITUDES OF THE SOLUTION  C
C  AT THE BEGINNING AND END OF A STEP.                            C
C                                                                 C
C  TO DISTINGUISH THE VARIOUS ARGUMENTS, H IS NOT PERMITTED TO    C
C  BECOME SMALLER THAN 26 UNITS OF ROUNDOFF IN T.                 C
C                                                                 C
C  PRACTICAL LIMITS ON THE CHANGE IN THE STEPSIZE ARE ENFORCED   C
C  TO SMOOTH THE STEPSIZE SELECTION PROCESS AND TO AVOID          C
C  EXCESSIVE CHATTERING ON PROBLEMS HAVING DISCONTINUITIES.       C
C                                                                 C
C  TO PREVENT UNNECESSARY FAILURES, THE CODE USES 9/10 THE        C
C  STEPSIZE IT ESTIMATES WILL SUCCEED.                            C
C                                                                 C
C  AFTER A STEP FAILURE, THE STEPSIZE IS NOT ALLOWED TO INCREASE C
C  FOR THE NEXT ATTEMPTED STEP.  THIS MAKES THE CODE MORE         C
C  EFFICIENT ON PROBLEMS HAVING DISCONTINUITIES AND MORE          C
C  EFFECTIVE IN GENERAL SINCE LOCAL EXTRAPOLATION IS BEING USED  C
C  AND EXTRA CAUTION SEEMS WARRANTED.                             C
C                                                                 C
C                                                                 C
C  TEST WHETHER THE NUMBER OF FUNCTION EVALUATIONS HAS EXCEEDED  C
C  THE ALLOWED LIMIT.                                             C
C                                                                 C
  200 IF (NFE.GT.MAXNFE) GO TO 500
C                                                                 C
C  ADVANCE AN APPROXIMATE SOLUTION OVER ONE STEP OF LENGTH H      C
C                                                                 C
C  THE FORMULAS HAVE BEEN GROUPED TO CONTROL LOSS OF              C
C  SIGNIFICANCE.                                                  C
C                                                                 C
      CH=0.25*H
      DO 221 I=1,NEQN
  221   W(I)=Y(I)+CH*YP(I)
      CALL F(T+0.25*H,W,F1)
C                                                                 C
      CH=0.09375*H
      DO 222 I=1,NEQN
  222   W(I)=Y(I)+CH*(YP(I)+3.*F1(I))
      CALL F(T+0.375*H,W,F2)
```

```
C                                                                      C
      CH=H/2197.
      DO 223 I=1,NEQN
  223   W(I)=Y(I)+CH*(1932.*YP(I)+(7296.*F2(I)-7200.*F1(I)))
      CALL F(T+12.*H/13.,W,F3)
C                                                                      C
      CH=H/4104.
      DO 224 I=1,NEQN
  224   W(I)=Y(I)+CH*((8341.*YP(I)-845.*F3(I))+
     1         (29440.*F2(I)-32832.*F1(I)))
      CALL F(T+H,W,F4)
C                                                                      C
      CH=H/20520.
      DO 225 I=1,NEQN
  225   W(I)=Y(I)+CH*((-6080.*YP(I)+(9295.*F3(I)-
     1          5643.*F4(I)))+(41040.*F1(I)-28352.*F2(I)))
      CALL F(T+0.5*H,W,F5)
      NFE=NFE+5
C                                                                      C
C  COMPUTE SOLUTION W AT T+H                                           C
C  COMPUTE SCALED RATIO EEOET OF LOCAL ERROR ESTIMATE TO               C
C    ALLOWABLE TOLERANCE.                                              C
C                                                                      C
      CH=H/7618050.
      EEOET=0.0
C                                                                      C
      DO 230 I=1,NEQN
C                                                                      C
        W(I)=Y(I)+CH*((902880.*YP(I)+(3855735.*F3(I)-
     1        1371249.*F4(I)))+(3953664.*F2(I)+277020.*F5(I)))
C                                                                      C
        ET=ABS(Y(I))+ABS(W(I))+AE
C                                                                      C
C     IF ET VANISHES, IT IS IMPOSSIBLE TO PROCEED.  THE USER           C
C     MUST PROVIDE A NONZERO VALUE OF ABSERR.                          C
C                                                                      C
        IF (ET.EQ.0.0) GO TO 480
C                                                                      C
        EE=ABS((-2090.*YP(I)+(21970.*F3(I)-15048.*F4(I)))+
     1                         (22528.*F2(I)-27360.*F5(I)))
  230   EEOET=AMAX1(EEOET,EE/ET)
C                                                                      C
C  REMOVE SCALING OF THE ERROR ESTIMATE RATIO.  THE STEP IS            C
C  ACCEPTED IF THE RESULT DOES NOT EXCEED 1.                           C
C                                                                      C
      ESTTOL = ABS(H)*EEOET*SCALE/752400.
      IF (ESTTOL.LE.1.) GO TO 260
C                                                                      C
C  THE STEP TO T+H DID NOT SUCCEED.                                    C
C    REDUCE THE STEPSIZE AND MAKE ANOTHER ATTEMPT.                     C
C    THE DECREASE IS LIMITED TO A FACTOR OF 1/10                       C
C                                                                      C
```

```
      HFAILD=.TRUE.
      OUTPUT=.FALSE.
      S=0.1
      IF (ESTTOL.LT.59049.) S=0.9/ESTTOL**0.2
      H=S*H
      IF (ABS(H).GE.HMIN) GO TO 200
C                                                                    C
C  THE REQUESTED ERROR TOLERANCES RELERR AND ABSERR ARE TOO          C
C  SMALL TO BE MET AT THE SMALLEST ALLOWABLE STEPSIZE.  THEY         C
C  ARE INCREASED TO VALUES WHICH SHOULD ALLOW SUCCESS.               C
C                                                                    C
      GFACT=(ABS(HMIN/H))**5
      IF (GFACT.LT.2.) GFACT=2.
      RELERR=GFACT*RELERR
      ABSERR=GFACT*ABSERR
      H=SIGN(HMIN,H)
      IFLAG=ISIGN(4,IFLAG)
      RETURN
C                                                                    C
C  THE STEP TO T+H WAS SUCCESSFUL.                                   C
C  IF THE PROBLEM HAS ALREADY BEEN FOUND STIFF, BYPASS FURTHER       C
C  TESTING.                                                          C
C                                                                    C
  260 IF (STIFF) GO TO 290
C                                                                    C
C  INCREMENT COUNTER OF SUCCESSFUL STEPS.  IF THE CURRENT BLOCK      C
C  OF 50 STEPS HAS ALREADY BEEN FOUND NOT STIFF, DO NOT TEST         C
C  THIS STEP.                                                        C
C                                                                    C
      NSS=NSS+1
      IF (NOTEST) GO TO 285
C                                                                    C
C  TEST WHETHER THIS STEP SHOWS STIFFNESS.                           C
C                                                                    C
      EECOET=0.0
      DO 275 I=1,NEQN
        ET=ABS(Y(I))+ABS(W(I))+AE
        EEC=ABS((-720895.*YP(I)+475423.*F2(I))+(461389.*F1(I)
     1      -300391.*F3(I))+123695.*F4(I)-39221.*F5(I))
  275   EECOET=AMAX1(EECOET,EEC/ET)
      IF (ABS(H)*EECOET*SCALE.GT.13.E6) GO TO 280
C                                                                    C
C  THIS STEP SHOWS STIFFNESS.  IF THE NUMBER OF SUCH STEPS IN        C
C  THE CURRENT BLOCK HAS REACHED 25, SET STIFF AND NOTEST TO         C
C  INDICATE THAT THE PROBLEM IS STIFF, AND TO BYPASS ALL             C
C  TESTING OF SUBSEQUENT STEPS.                                      C
C                                                                    C
      KST=KST+1
      IF (KST.LT.25) GO TO 285
      STIFF=.TRUE.
      NOTEST=.TRUE.
      GO TO 290
```

```
C                                                                      C
C  THIS STEP DOES NOT SHOW STIFFNESS.  IF THE CURRENT BLOCK IS         C
C  THEREBY ESTABLISHED AS NOT STIFF, SET NOTEST=.TRUE. TO              C
C  SUSPEND TESTING FOR THE REMAINDER OF THIS BLOCK.                    C
C                                                                      C
  280 IF ((NSS-KST).GT.25) NOTEST=.TRUE.
C                                                                      C
C  THE PROBLEM HAS NOT BEEN ESTABLISHED AS STIFF.  IF THE              C
C  CURRENT BLOCK IS COMPLETE,  REINITIALIZE INDICATORS TO              C
C  TEST NEXT BLOCK.                                                    C
C                                                                      C
  285 IF (NSS.LT.50) GO TO 290
      NSS=0
      KST=0
      NOTEST=.FALSE.
C                                                                      C
C  REPLACE T WITH T+H                                                  C
C  STORE SOLUTION IN Y                                                 C
C  EVALUATE DERIVATIVES YP                                             C
C                                                                      C
  290 T=T+H
      DO 295 K=1,NEQN
  295   Y(K)=W(K)
      CALL F(T,Y,YP)
      NFE=NFE+1
C                                                                      C
C  ESTIMATE OPTIMUM STEPSIZE FOR NEXT STEP.                            C
C  THE INCREASE IS LIMITED TO A FACTOR OF 5                            C
C  IF STEP FAILURE HAS JUST OCCURRED, THE STEPSIZE IS NOT              C
C  ALLOWED TO INCREASE.                                                C
C                                                                      C
      S=5.0
      IF (ESTTOL.GT.1.889568E-4) S=0.9/ESTTOL**0.2
      IF (HFAILD.AND.(S.GT.1.0)) S=1.0
      H=SIGN(AMAX1(S*ABS(H),HMIN),H)
C                                                                      C
C**********************************************************************C
C  END OF CORE INTEGRATOR                                              C
C**********************************************************************C
C                                                                      C
C  DETERMINE WHETHER TO TAKE ANOTHER STEP BEFORE RETURN.               C
C                                                                      C
      IF (OUTPUT) GO TO 300
      IF (IFLAG.GT.0) GO TO 100
C                                                                      C
C**********************************************************************C
C**********************************************************************C
C                                                                      C
C  INTEGRATION SUCCESSFULLY COMPLETED                                  C
C                                                                      C
C     SINGLE-STEP MODE                                                 C
C                                                                      C
```

```
      IFLAG=-2
      RETURN
C                                                                     C
C    INTERVAL MODE                                                    C
C                                                                     C
  300 T=TOUT
      IFLAG=2
      RETURN
C                                                                     C
C**********************************************************************C
C**********************************************************************C
C                                                                     C
C  INITIALIZATION SECTION --                                          C
C                                                                     C
C                                                                     C
C  ON FIRST CALL (IFLAG = +1 OR -1),                                  C
C    SET INITIALIZATION COMPLETION INDICATOR INIT                     C
C    SET EXCESSIVE OUTPUT FREQUENCY INDICATOR KOP                     C
C    SET FUNCTION EVALUATION COUNTER NFE                              C
C    SET STIFFNESS TEST COUNTERS NSS,KST                              C
C    EVALUATE INITIAL DERIVATIVES                                     C
C    IF TOUT = T, RETURN TO CALLING PROGRAM                           C
C                                                                     C
  400 INIT=0
      KOP=0
      NFE=1
      NSS=0
      KST=0
      CALL F(T,Y,YP)
      IF (T.NE.TOUT) GO TO 410
      IFLAG=2
      RETURN
C                                                                     C
C  TEST WHETHER REQUESTED ERROR TOLERANCE IS TOO SMALL FOR THE        C
C  MACHINE PRECISION.                                                 C
C                                                                     C
  410 IF (RELERR.LT.REMIN) GO TO 440
C                                                                     C
C  ESTIMATE STARTING STEPSIZE, SET INIT = 1 TO INDICATE THAT          C
C  THIS HAS BEEN DONE, AND TRANSFER TO BEGIN INTEGRATION.             C
C                                                                     C
      INIT=1
      H=ABS(DT)
      TOLN=0.0
      DO 420 K=1,NEQN
        TOL=RELERR*ABS(Y(K))+ABSERR
        IF (TOL.LE.0.0) GO TO 420
        TOLN=TOL
        YPK=ABS(YP(K))
        IF (YPK*H**5.GT.TOL) H=(TOL/YPK)**0.2
  420 CONTINUE
      IF (TOLN.LE.0.0) H=0.0
      H=AMAX1(H,U26*AMAX1(ABS(T),ABS(DT)))
      GO TO 80
C                                                                C
C****************************************************************C
C****************************************************************C
C                                                                C
C  THE REMAINDER OF THE PROGRAM CONTAINS STATEMENTS TO DEAL WITH C
C  VARIOUS EXCEPTIONAL OCCURRENCES.                              C
C                                                                C
C                                                                C
C  INVALID INPUT WAS PASSED TO RKFS.                             C
C                                                                C
  430 IFLAG=9
      RETURN
C                                                                C
C  RELERR IS TOO SMALL FOR THE MACHINE PRECISION, AND IS RESET   C
C  TO THE MINIMUM PERMISSIBLE VALUE.                             C
C                                                                C
  440 RELERR=REMIN
      IFLAG=ISIGN(3,IFLAG)
      RETURN
C                                                                C
C UNNECESSARY FREQUENCY OF OUTPUT                                C
C                                                                C
  450 KOP=0
      IFLAG=ISIGN(7,IFLAG)
      RETURN
C                                                                C
C  RKFS WAS CALLED WITH TOUT CLOSER TO T THAN THE MINIMUM        C
C  ALLOWABLE STEPSIZE.  EXTRAPOLATE BY EULER METHOD AND RETURN.  C
C                                                                C
  460 DO 470 K=1,NEQN
  470   Y(K)=Y(K)+DT*YP(K)
      CALL F(TOUT,Y,YP)
      NFE=NFE+1
      T=TOUT
      IFLAG=2
      RETURN
C                                                                C
C  THE ERROR TOLERANCE FUNCTION ET VANISHED, MAKING IT           C
C  IMPOSSIBLE TO APPLY THE LOCAL ERROR TEST.  TO AVOID THIS,     C
C  THE USER MUST SUPPLY A NONZERO ABSERR.                        C
C                                                                C
  480   IFLAG=8
        RETURN
C                                                                C
C  NUMBER OF FUNCTION CALLS HAS EXCEEDED MAXNFE.  SET FLAG TO    C
C  INDICATE WHETHER THIS RESULTS FROM STIFFNESS.                 C
C                                                                C
  500 IFLAG=ISIGN(5,IFLAG)
      NFE=NFE-MAXNFE
      NSS=0
      KST=0
      IF (STIFF) IFLAG=ISIGN(6,IFLAG)
      RETURN
C                                                                      C
C**********************************************************************C
C**********************************************************************C
C                                                                      C
C  END OF RKFS                                                         C
C                                                                      C
      END
```

APPENDIX B

ADAMS METHOD PROGRAMS

This appendix contains the complete listings of the subroutines ADMINT, ADAM, STEP, and SINTRP.

```
      SUBROUTINE ADMINT(NEQN,Y,W,TINIT,TFINAL,TINCR,RELERR,
     1                   ABSERR,IUNIT,FCT,OUTP)
C                                                                    C
C********************************************************************C
C                                                                    C
C  SUBROUTINE ADMINT USES ADAM TO INTEGRATE A SYSTEM OF NEQN        C
C  FIRST ORDER ORDINARY DIFFERENTIAL EQUATIONS                       C
C                                                                    C
C    DY/DT = F(T,Y) ,  Y = (Y(1),Y(2),...,Y(NEQN))                   C
C                                                                    C
C  FROM TINIT TO TFINAL, WITH OUTPUT OF THE SOLUTION                C
C  AT INTERVALS OF TINCR BY MEANS OF CALLS TO THE USER-             C
C  -SUPPLIED SUBROUTINE OUTP.                                        C
C                                                                    C
C                                                                    C
C  THE QUANTITIES IN THE ARGUMENT LIST REPRESENT                     C
C                                                                    C
C    NEQN -- THE NUMBER OF EQUATIONS                                 C
C    Y -- AN ARRAY OF DIMENSION NEQN, WHICH CONTAINS THE            C
C         VALUE OF THE SOLUTION                                      C
C    W -- AN ARRAY OF DIMENSION 19*NEQN, WHICH IS USED FOR         C
C         WORKING STORAGE.                                           C
C    TINIT -- THE STARTING VALUE OF THE INDEPENDENT VARIABLE       C
C    TFINAL -- THE VALUE OF THE INDEPENDENT VARIABLE AT WHICH    C
C               INTEGRATION IS TO STOP                               C
C    TINCR -- THE DESIRED SPACING, IN THE INDEPENDENT VARIABLE,     C
C              OF THE OUTPUT POINTS                                  C
C    RELERR -- THE DESIRED RELATIVE ERROR TOLERANCE                 C
C    ABSERR -- THE DESIRED ABSOLUTE ERROR TOLERANCE                 C
C    IUNIT -- THE LOGICAL UNIT NUMBER OF THE DATA SET TO BE         C
C              USED FOR WRITING DIAGNOSTIC MESSAGES.  IF IUNIT      C
C              IS LESS THAN OR EQUAL TO ZERO, NO MESSAGES ARE       C
C              WRITTEN                                               C
C    FCT -- THE NAME OF THE USER-SUPPLIED SUBROUTINE                C
C            FCT(T,Y,YP) WHICH EVALUATES THE DERIVATIVES            C
C            YP(I)=DY(I)/DT, I=1,2,...,NEQN                          C
C    OUTP -- THE NAME OF THE USER-SUPPLIED SUBROUTINE               C
```

```
C             OUTP(T,Y,W) USED TO HANDLE OUTPUT.                       C
C                                                                      C
C                                                                      C
C  TO USE ADMINT, THE CALLING PROGRAM MUST                             C
C                                                                      C
C    1. ALLOCATE SPACE FOR THE ARRAYS Y(*) AND W(*) IN A               C
C       DIMENSION STATEMENT.  (NOTE THAT Y MUST BE AN ARRAY,           C
C       EVEN IF NEQN=1.)                                               C
C                                                                      C
C    2. SET Y(1),Y(2),...,Y(NEQN) TO THE INITIAL VALUES TO BE          C
C       USED FOR THE SOLUTION.                                         C
C                                                                      C
C    3. PROVIDE APPROPRIATE VALUES FOR NEQN, TINIT, TFINAL,            C
C       TINCR, RELERR, ABSERR, AND IUNIT.  (SINCE THESE                C
C       QUANTITIES ARE NOT ALTERED BY ADMINT, THEY MAY, IF             C
C       DESIRED, BE WRITTEN AS CONSTANTS IN THE CALL LIST.)            C
C                                                                      C
C    4. PLACE THE ACTUAL NAMES OF FCT AND OUTP IN THE CALL             C
C       LIST, AND DECLARE THOSE NAMES IN AN EXTERNAL STATEMENT.        C
C                                                                      C
C                                                                      C
C                                                                      C
C  AT EACH CALL TO OUTP(T,Y,W), THE CURRENT VALUES OF THE              C
C  DERIVATIVES DY(I)/DT ARE IN W(1),W(2),...,W(NEQN).  THE             C
C  FIRST CALL TO OUTP IS MADE AT TINIT.                                C
C                                                                      C
C  IT IS NOT NECESSARY THAT THE INTEGRATION INTERVAL BE EVENLY         C
C  DIVISIBLE BY THE OUTPUT MESH SIZE TINCR.  HOWEVER, IF IT IS         C
C  NOT, INTEGRATION WILL PROCEED TO THE FIRST MESH POINT               C
C  BEYOND TFINAL, RATHER THAN STOPPING AT TFINAL.                      C
C                                                                      C
C**********************************************************************C
C                                                                      C
C  THIS SUBROUTINE IS WRITTEN IN SINGLE PRECISION.                     C
C  A DOUBLE PRECISION VERSION FOR USE WITH A DOUBLE PRECISION          C
C  VERSION OF ADAM MAY BE OBTAINED SIMPLY BY REMOVING THE C            C
C  FROM COLUMN 1 OF THE NEXT FIVE CARDS.                               C
C     DOUBLE PRECISION ABS,ABSER,ABSERR,DABS,DSIGN,DT,DTSGN,
C    1                 D1,D2,EPSDT,RELER,RELERR,SIGN,T,TFINAL,
C    2                 TINCR,TINIT,TMAXIN,TOUT,W,Y
C     ABS(D1) = DABS(D1)
C     SIGN(D1,D2) = DSIGN(D1,D2)
C                                                                      C
      COMMON /ADDATA/  T,TOUT,RELER,ABSER,DTSGN,NEQ,IFLAG,
     1                 IOLD,KLE4,INIT,KOP
C                                                                      C
      DIMENSION Y(NEQN),W(1)
C                                                                      C
C  THE ACTUAL DIMENSION OF W, AS DEFINED BY THE CALLING PROGRAM,       C
C  MUST BE AT LEAST 19*NEQN.                                           C
C                                                                      C
C                                                                      C
```

```
      EXTERNAL FCT
C                                                                C
C  WRITE STARTING DATA INTO MESSAGE FILE                         C
C                                                                C
      IF (IUNIT.GT.0) WRITE(IUNIT,100) NEQN,TINIT,TFINAL,TINCR,
     1                                           RELERR,ABSERR
C                                                                C
C  COMPUTE INDICES FOR SPLITTING WORK ARRAY W(*).                C
C                                                                C
      K1=NEQN+1
      K2=K1+NEQN
      K3=K2+NEQN
C                                                                C
C  INITIALIZE VARIABLES TO START INTEGRATION WITH ADAM.          C
C                                                                C
      NEQ=NEQN
      RELER=RELERR
      ABSER=ABSERR
      IFLAG=1
      T=TINIT
      TOUT=TINIT
C                                                                C
C  SAVE INFORMATION NEEDED TO LOCATE END OF INTEGRATION.         C
C                                                                C
      TMAXIN=ABS(TFINAL-TINIT)
      EPSDT=0.001*ABS(TINCR)
C                                                                C
C  SET SIGN OF OUTPUT INTERVAL TO PROCEED FROM TINIT TO TFINAL.  C
C                                                                C
      DT=SIGN(TINCR,TFINAL-TINIT)
C                                                                C
C  INITIALIZE COUNTER OF ERROR RETURNS.                          C
C                                                                C
      IER=0
C                                                                C
C****************************************************************C
C                                                                C
C  CALL ADAM TO INTEGRATE TO THE NEXT OUTPUT POINT.              C
C                                                                C
    5 CALL ADAM(FCT,Y,W(1),W(K1),W(K2),W(K3))
      IF (IFLAG.NE.2) GO TO 8
C                                                                C
C  SUCCESSFUL INTEGRATION TO OUTPUT POINT.                       C
C                                                                C
      CALL OUTP(T,Y,W)
C                                                                C
C  CHECK WHETHER TFINAL HAS BEEN REACHED OR PASSED.              C
C                                                                C
      IF (ABS(T-TINIT).GE.TMAXIN) GO TO 95
C                                                                C
C  INTEGRATION IS NOT YET COMPLETE, SO THE NEXT OUTPUT POINT IS  C
C  SET.  IF THE NEW POINT DIFFERS FROM TFINAL BY AN AMOUNT       C
```

```
C  ATTRIBUTABLE TO ROUNDOFF ERROR, IT IS RESET TO TFINAL EXACTLY.C
C  INTEGRATION IS THEN CONTINUED.                                C
C                                                                C
      TOUT=T+DT
      IF (ABS(TFINAL-TOUT).LT.EPSDT) TOUT=TFINAL
      GO TO 5
C                                                                C
C****************************************************************C
C                                                                C
C  INTEGRATION WAS INTERRUPTED BEFORE REACHING THE OUTPUT POINT. C
C  INCREMENT THE ERROR COUNTER AND BRANCH FOR WRITING THE        C
C  APPROPRIATE DIAGNOSTIC MESSAGE.  INTEGRATION WILL THEN BE     C
C  CONTINUED IF THE ERROR IS NOT TOO SEVERE.                     C
C                                                                C
    8 IER=IER+1
      GO TO (20,20,30,40,50,60,70,80,90),IFLAG
C                                                                C
C                                                                C
C  IFLAG HAS AN ILLEGAL VALUE, SO THAT A PRECISE DEFINITION OF   C
C  THE ERROR IS NOT POSSIBLE, AND EXECUTION MUST BE TERMINATED.  C
C                                                                C
   20 IF (IUNIT.GT.0) WRITE(IUNIT,200) T,IFLAG
      STOP
C                                                                C
C                                                                C
C  IFLAG = 3 -- THE REQUESTED ERROR TOLERANCES WERE TOO          C
C  SMALL FOR THE MACHINE PRECISION AND HAVE BEEN INCREASED TO    C
C  MINIMUM PERMISSIBLE VALUES.                                   C
C                                                                C
   30 IF (IUNIT.GT.0) WRITE(IUNIT,300) T,RELER,ABSER
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C  IFLAG = 4 -- THE REQUESTED ERROR TOLERANCES WERE TOO          C
C  STRINGENT FOR THIS PROBLEM AND HAVE BEEN INCREASED TO         C
C  VALUES WHICH SHOULD BE ACCEPTABLE.                            C
C                                                                C
   40 IF (IUNIT.GT.0) WRITE(IUNIT,400) T,RELER,ABSER
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 5 -- THE COUNTER OF CALLS TO FCT WAS RESET AFTER      C
C  EXCEEDING THE LIMIT IMPOSED IN ADAM.                          C
C                                                                C
   50 IF (IUNIT.GT.0) WRITE(IUNIT,500) T
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 6 -- THE COUNTER OF CALLS TO FCT WAS RESET AFTER      C
C  STIFFNESS CAUSED IT TO EXCEED THE LIMIT IMPOSED IN ADAM.      C
```

2 OF

PB

190

OF 2

880

986

1.0
4.5
2.8
2.5
5.0
5.6
3.2
2.2
6.3
3.6
7.1
8.0
4.0
2.0
1.1
1.8
.25
1.4
1.6

```
C                                                                C
   60 IF (IUNIT.GT.0) WRITE(IUNIT,600) T
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 7 -- THE MONITOR OF OUTPUT FREQUENCY WAS RESET        C
C  AFTER INDICATING THAT THE OUTPUT INTERVAL IS TOO SMALL        C
C  TO ALLOW EFFICIENT OPERATION OF ADAM.                         C
C                                                                C
   70 IF (IUNIT.GT.0) WRITE(IUNIT,700) T
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 8 -- FATAL ERROR -- A COMPONENT OF THE SOLUTION       C
C  VANISHED, MAKING A PURE RELATIVE ERROR TEST IMPOSSIBLE.       C
C  THE USER MUST SUPPLY AN APPROPRIATE NONZERO VALUE OF          C
C  ABSERR FOR SUCCESSFUL INTEGRATION.                            C
C                                                                C
   80 IF (IUNIT.GT.0) WRITE(IUNIT,800) T
      STOP
C                                                                C
C                                                                C
C  IFLAG = 9 -- FATAL ERROR -- AN ILLEGAL PARAMETER VALUE        C
C  WAS PASSED TO ADAM.  THE USER MUST LOCATE AND CORRECT THE     C
C  ERROR.                                                        C
C                                                                C
   90 IF (IUNIT.GT.0) WRITE(IUNIT,900) T,NEQ,RELER,ABSER,TINCR
      STOP
C                                                                C
C                                                                C
C  THE PROGRAM IS STOPPED BECAUSE THERE HAVE BEEN FIVE ERROR     C
C  RETURNS FROM ADAM, INDICATING SERIOUS PROBLEMS WHICH          C
C  REQUIRE INTERVENTION BY THE USER.                             C
C                                                                C
   92 IF (IUNIT.GT.0) WRITE(IUNIT,920) T
      STOP
C                                                                C
C                                                                C
C  THE INTEGRATION HAS BEEN CONPLETED TO OR BEYOND TFINAL.       C
C                                                                C
   95 IF (IUNIT.GT.0) WRITE(IUNIT,950) T
      RETURN
C                                                                C
C****************************************************************C
C                                                                C
C  THE REMAINDER OF THE PROGRAM CONTAINS THE FORMAT STATEMENTS   C
C  FOR THE IDAGNOSTIC MESSAGES.                                  C
C                                                                C
  100 FORMAT(36H ADMINT HAS BEEN CALLED TO INTEGRATE,I5,
     1  10H EQUATIONS/18H FROM INITIAL TIME,G12.5/
     2  14H TO FINAL TIME,G12.5/
     3  24H WITH OUTPUT AT STEPS OF,G12.5/
     4  32H AND INITIAL ERROR TOLERANCES OF/
     5  9H RELERR =,G12.5,9H ABSERR =,G12.5/)
C                                                                C
  200 FORMAT(4H ***,12(4H****)/
     1  37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2  52H *** BECAUSE AN UNIDENTIFIABLE ERROR HAS CAUSED  ***/
     3  52H *** ADAM TO RETURN WITH THE ILLEGAL FLAG VALUE  ***/
     4  12H *** IFLAG =,I11,26X,3H***/4H ***,12(4H****))
C                                                                C
  300 FORMAT(17H *WARNING--AT T =,G12.5,22H, THE ERROR TOLERANCES/
     1  50H PASSED TO ADAM WERE FOUND TO BE TOO SMALL FOR THE/
     2  48H MACHINE PRECISION.  THEY HAVE BEEN INCREASED TO/
     3  9H RELERR =,G12.5,13H AND ABSERR =,G12.5/
     4  25H TO CONTINUE INTEGRATION./)
C                                                                C
  400 FORMAT(17H *WARNING--AT T =,G12.5,20H, ADAM WAS UNABLE TO/
     1  52H MEET THE REQUESTED ERROR TOLERANCES AT THE SMALLEST/
     2  52H ALLOWABLE STEPSIZE.  INTEGRATION IS CONTINUING WITH/
     3  37H THE TOLERANCES INCREASED TO RELERR =,G12.5/
     4  13H AND ABSERR =,G12.5/)
C                                                                C
  500 FORMAT(17H *WARNING--AT T =,G12.5,24H, THE NUMBER OF FUNCTION/
     1  51H CALLS EXCEEDED THE MAXIMUM NUMBER, MAXNFE, ALLOWED/
     2  51H BY ADAM.  THE FUNCTION EVALUATION COUNTER HAS BEEN/
     3  51H REDUCED BY MAXNFE TO ALLOW INTEGRATION TO PROCEED./)
C                                                                C
  600 FORMAT(17H *WARNING--AT T =,G12.5,22H, STIFFNESS HAS CAUSED/
     1  51H THE NUMBER OF FUNCTION CALLS TO EXCEED THE MAXIMUM/
     2  47H NUMBER, MAXNFE, ALLOWED BY ADAM.  THE FUNCTION/
     3  49H EVALUATION COUNTER HAS BEEN REDUCED BY MAXNFE TO/
     4  30H ALLOW INTEGRATION TO PROCEED./)
C                                                                C
  700 FORMAT(17H *WARNING--AT T =,G12.5,23H, ADAM RETURNED WITH AN/
     1  50H INDICATION THAT THIS INTEGRATION IS UNNECESSARILY/
     2  48H COSTLY BECAUSE THE REQUESTED OUTPUT INTERVAL IS/
     3  49H SUBSTANTIALLY SMALLER THAN THE NATURAL STEPSIZE./
     4  48H (REFER TO PROGRAM DOCUMENTATION FOR ALTERNATIVE/
     5  52H APPROACHES.)  THE OUTPUT FREQUENCY MONITOR HAS BEEN/
     6  39H RESET TO ALLOW INTEGRATION TO PROCEED./)
C                                                                C
  800 FORMAT(4H ***,12(4H****)/
     1  37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2  40H *** BECAUSE A COMPONENT OF THE SOLUTION,9X,3H***/
     3  52H *** VANISHED WHILE ABSERR = 0.  IT WILL BE       ***/
     4  52H *** NECESSARY TO PROVIDE A NONZERO ABSOLUTE      ***/
     5  52H *** ERROR TOLERANCE TO PERFORM THIS INTEGRATION.***/
     6  4H ***,12(4H****))
C                                                                C
  900 FORMAT(4H ***,12(4H****)/
     1  37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2  52H *** BECAUSE INVALID INPUT WAS RECEIVED BY ADAM. ***/
     3  37H *** CHECK THE FOLLOWING PARAMETERS--,12X,3H***/
     4  12H ***  NEQN =,I11,26X,3H***/
     5  14H ***  RELERR =,G12.5,23X,3H***/
     6  14H ***  ABSERR =,G12.5,23X,3H***/
     7  13H ***  TINCR =,G12.5,24X,3H***/
     8  52H *** ALSO, CHECK THAT THE OUTPUT ROUTINE DOES     ***/
     9  41H *** NOT RESET IFLAG TO AN ILLEGAL VALUE.,8X,3H***/
     A  4H ***,12(4H****))
C                                                                C
  920 FORMAT(4H ***,12(4H****)/
     1  37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2  52H *** BECAUSE ADAM HAS RETURNED IRREGULARLY FIVE  ***/
     3  52H *** TIMES, WHICH INDICATES SOME SORT OF EXTREME ***/
     4  52H *** DIFFICULTY IN THE INTEGRATION.  THE USER    ***/
     5  52H *** SHOULD CAREFULLY CHECK THE ACCURACY OF HIS  ***/
     6  52H *** PROBLEM STATEMENT AND CODING.  IT IS ALSO   ***/
     7  52H *** POSSIBLE THAT THIS PROBLEM IS TOO STIFF TO  ***/
     8  27H *** BE INTEGRATED BY ADAM.,22X,3H***/
     9  4H ***,12(4H****))
C                                                                C
  950 FORMAT(4H ***,12(4H****)/
     1  32H *** INTEGRATION COMPLETE AT T =,G12.5,5X,3H***/
     2  4H ***,12(4H****)///)
C                                                                C
C****************************************************************C
C                                                                C
C  END OF ADMINT                                                 C
C                                                                C
      END


      SUBROUTINE ADAM(F,Y,YP,WT,YY,PHI)
C                                                                C
C****************************************************************C
C                                                                C
C  SUBROUTINE ADAM USES SUBROUTINE STEP TO INTEGRATE A YSYTEM OF C
C  NEQN FIRST ORDER ORDINARY DIFFERENTIAL EQUATIONS              C
C                                                                C
C    DY/DT = F(T,Y) ,  Y = (Y(1),Y(2),...,Y(NEQN))               C
C                                                                C
C  FROM T TO TOUT.  FOR REASONS OF EFFICIENCY, ADAM INTEGRATES   C
C  BEYOND TOUT INTERNALLY, THOUGH NEVER BEYOND T + 10.*(TOUT-T), C
C  AND CALLS SUBROUTINE SINTRP TO INTERPOLATE THE SOLUTION AT    C
C  TOUT.  AN OPTION IS PROVIDED TO STOP THE INTEGRATION AT TOUT, C
C  BUT IT SHOULD BE USED ONLY IF THE CHARACTERISTICS OF THE      C
C  PROBLEM MAKE IT IMPOSSIBLE TO CONTINUE THE INTEGRATION BEYOND C
C  TOUT.                                                         C
C                                                                C
C  THE PARAMETERS IN THE ARGUMENT LIST ARE                       C
C                                                                C
C    F -- THE NAME OF THE USER-SUPPLIED SUBROUTINE F(T,Y,YP),    C
```

```
C         WHICH EVALUATES THE DERIVATIVES YP(I)=DY(I)/DT,        C
C         I=1,2,...,NEQN.                                        C
C    Y -- AN ARRAY OF DIMENSION NEQN, WHICH CONTAINS THE VALUE   C
C         OF THE SOLUTION.                                       C
C    YP -- AN ARRAY OF DIMENSION NEQN, WHICH CONTAINS THE VALUE  C
C          OF THE DERIVATIVE DY/DT.                              C
C    WT,YY,PHI -- ARRAYS OF DIMENSIONS NEQN, NEQN, AND 16*NEQN,  C
C                  RESPECTIVELY, WHICH ARE USED FOR WORKING      C
C                  STORAGE BY ADAM AND STEP.  THE VALUES STORED  C
C                  IN YY AND PHI ARE NEEDED FOR CONTINUING       C
C                  INTEGRATION ON SUBSEQUENT CALLS.              C
C                                                                C
C  THE PARAMETERS IN THE COMMON BLOCK ADDATA ARE                 C
C                                                                C
C    T -- THE INDEPENDENT VARIABLE                               C
C                                                                C
C    TOUT -- THE VALUE OF T AT WHICH OUTPUT IS DESIRED           C
C                                                                C
C    RELERR,ABSERR -- RELATIVE AND ABSOLUTE ERROR TOLERANCES FOR C
C       THE LOCAL ERROR TEST.  EACH STEPSIZE IS CHOSEN SO THAT   C
C       ABS(LOCAL ERROR(I)) .LE. RELERR*ABS(Y(I))+ABSERR FOR     C
C       EACH COMPONENT OF THE SOLUTION.                          C
C                                                                C
C    NEQN -- THE NUMBER OF EQUATIONS                             C
C                                                                C
C    IFLAG -- INDICATOR OF THE STATUS OF INTEGRATION             C
C                                                                C
C    DTSGN,IOLD,KLE4,INIT,KOP -- THOSE QUANTITIES USED FOR       C
C       INTERNAL BOOKKEEPING BY ADAM WHICH ARE NEEDED FOR        C
C       CONTINUING INTEGRATION, BUT WHICH ARE NORMALLY OF NO     C
C       INTEREST TO THE USER.  THEY ARE PLACED IN THE COMMON     C
C       BLOCK TO AVOID LOCAL RETENTION BY ADAM BETWEEN CALLS.    C
C       SEE THE PROGRAM DOCUMENTATION FOR A DESCRIPTION OF THEIR C
C       FUNCTIONING.                                             C
C                                                                C
C----------------------------------------------------------------C
C                                                                C
C  TO USE ADAM, THE CALLING PROGRAM MUST                         C
C                                                                C
C    1. ALLOCATE SPACE FOR THE ARRAYS Y(NEQN), YP(NEQN),         C
C       WT(NEQN), YY(NEQN), AND PHI(NEQN,16).                    C
C       (Y, YP, WT, AND YY MUST BE ARRAYS EVEN IF NEQN=1.)       C
C                                                                C
C    2. DECLARE THE NAME OF F IN AN EXTERNAL STATEMENT, AND      C
C       PLACE THAT NAME IN THE CALL LIST.                        C
C                                                                C
C                                                                C
C  FIRST CALL TO ADAM                                            C
C  ------------------                                            C
C                                                                C
C  TO BEGIN INTEGRATION, THE CALLING PROGRAM MUST                C
C                                                                C
```

```
C    1. INITIALIZE THE FOLLOWING VARIABLES IN THE COMMON BLOCK   C
C       ADDATA                                                   C
C         T -- THE STARTING VALUE OF THE INDEPENDENT VARIABLE    C
C         TOUT -- THE VALUE OF THE INDEPENDENT VARIABLE AT WHICH C
C            OUTPUT IS DESIRED.  (TOUT=T IS PERMITTED ON THE     C
C            FIRST CALL ONLY, IN WHICH CASE ADAM EVALUATES THE   C
C            DERIVATIVES YP AND RETURNS WITH IFLAG=2.)           C
C         RELERR -- THE DESIRED RELATIVE ERROR TOLERANCE         C
C         ABSERR -- THE DESIRED ABSOLUTE ERROR TOLERANCE         C
C         IFLAG -- NORMALLY +1.  IF THE INTEGRATION CANNOT       C
C                  PROCEED BEYOND TOUT, SET IFLAG = -1.          C
C                                                                C
C    2. INITIALIZE Y(*) TO CONTAIN THE STARTING VALUE OF THE     C
C       SOLUTION.                                                C
C                                                                C
C                                                                C
C  UPON SUCCESSFUL RETURN (IFLAG=2), T=TOUT, Y CONTAINS THE      C
C  SOLUTION AT TOUT, AND YP CONTAINS THE DERIVATIVES AT TOUT.    C
C                                                                C
C  OTHERWISE, T IS THE LAST VALUE OF THE INDEPENDENT VARIABLE    C
C  SUCCESSFULLY REACHED DURING THE INTEGRATION, Y CONTAINS THE   C
C  SOLUTION AT T, YP CONTAINS THE DERIVATIVES AT T, AND IFLAG    C
C  INDICATES THE STATUS OF THE INTEGRATION AS DESCRIBED BELOW.   C
C                                                                C
C                                                                C
C  SUBSEQUENT CALLS TO ADAM                                      C
C  ------------------------                                      C
C                                                                C
C  ADAM NORMALLY RETURNS WITH ALL INFORMATION NEEDED TO CONTINUE C
C  INTEGRATION.                                                  C
C                                                                C
C  IF THE INTEGRATION REACHED TOUT (IFLAG=2), THE USER NEED      C
C  ONLY DEFINE A NEW TOUT AND CALL ADAM AGAIN.  IF THE INTE-     C
C  GRATION CANNOT CONTINUE INTERNALLY BEYOND THE NEW TOUT,       C
C  THE USER NUST ALSO SET IFLAG=-2.  IF THE NEW TOUT REPRESENTS  C
C  A REVERSAL OF THE DIRECTION OF INTEGRATION, IT IS NECESSARY   C
C  THAT Y NOT HAVE BEEN ALTERED.                                 C
C                                                                C
C  IF THE INTEGRATION WAS INTERRUPTED WITH ABS(IFLAG) = 3, 4, 5, C
C  6, OR 7, INTEGRATION MAY BE CONTINUED SIMPLY BY CALLING ADAM  C
C  AGAIN.  HOWEVER, SUCH A RETURN INDICATES THAT THE             C
C  INTEGRATION IS NOT PROCEEDING SO SMOOTHLY AS MIGHT BE         C
C  EXPECTED, SO THE USER SHOULD EXERCISE SOME CAUTION.           C
C                                                                C
C  IF THE INTEGRATION WAS INTERRUPTED WITH IFLAG = 8, THE USER   C
C  MUST GIVE ABSERR A NONZERO POSITIVE VALUE AND SET IFLAG = 2   C
C  (OR IFLAG=-2 IF IT IS NECESSARY TO STOP AT TOUT) IN ORDER TO  C
C  CONTINUE.                                                     C
C                                                                C
C  IF ADAM RETURNED WITH IFLAG = 9, THE USER MUST CORRECT THE    C
C  INVALID INPUT AND SET IFLAG = 2 OR -2 (OR IFLAG = 1 OR -1,    C
C  IF INTEGRATION HAS NOT YET STARTED) IN ORDER TO CONTINUE.     C
```

```
C                                                                C
C----------------------------------------------------------------C
C                                                                C
C  UPON RETURN, THE VALUE OF IFLAG INDICATES THE STATUS OF THE   C
C  INTEGRATION AS FOLLOWS,                                       C
C                                                                C
C  IFLAG = 2 -- T=TOUT, SO THAT INTEGRATION WAS SUCCESSFUL TO    C
C               THE OUTPUT POINT.                                C
C  IFLAG = 3 -- INTEGRATION WAS INTERRUPTED BECAUSE THE          C
C               REQUESTED ERROR TOLERANCES WERE TOO SMALL FOR    C
C               THE MACHINE PRECISION.  THEY HAVE BEEN INCREASED C
C               TO MINIMUM ACCEPTABLE VALUES.                    C
C  IFLAG = 4 -- INTEGRATION WAS INTERRUPTED BECAUSE THE          C
C               REQUESTED ERROR TOLERANCES COULD NOT BE          C
C               ATTAINED AT THE SMALLEST ALLOWABLE STEPSIZE.     C
C               THE TOLERANCES HAVE BEEN INCREASED TO VALUES     C
C               WHICH SHOULD ALLOW SUCCESSFUL CONTINUATION.      C
C  IFLAG = 5 -- INTEGRATION WAS INTERRUPTED AFTER THE COUNTER    C
C               OF CALLS TO F EXCEEDED MAXNFE.  THE COUNTER HAS  C
C               BEEN REDUCED BY MAXNFE.                          C
C  IFLAG = 6 -- SAME AS IFLAG=5, EXCEPT THAT THE PROBLEM HAS     C
C               BEEN ADDITIONALLY DIAGNOSED AS STIFF.            C
C  IFLAG = 7 -- INTEGRATION WAS NOT CONTINUED BECAUSE THE        C
C               SPACING OF OUTPUT POINTS HAS BEEN SO SMALL AS TO C
C               RESTRICT THE STEPSIZE FOR 50 CALLS TO STEP,      C
C               INDICATING THAT THE CODE IS BEING USED IN A WAY  C
C               WHICH GREATLY IMPAIRS ITS EFFICIENCY.  THE       C
C               MONITOR OF THIS EFFECT WAS RESET TO ALLOW        C
C               CONTINUATION IF DESIRED.                         C
C  IFLAG = 8 -- INTEGRATION WAS INTERRUPTED WHEN A COMPONENT OF  C
C               THE SOLUTION VANISHED WHILE A PURE RELATIVE      C
C               ERROR TEST WAS SPECIFIED.  THE USER MUST SUPPLY  C
C               A NONZERO ABSERR TO CONTINUE INTEGRATION.        C
C  IFLAG = 9 -- INVALID INPUT WAS PASSED TO ADAM.  THE USER MUST C
C               LOCATE AND CORRECT THE ERROR IN ORDER TO         C
C               CONTINUE INTEGRATION.                            C
C  IFLAG =-3, -4, -5, -6, OR -7 -- SAME AS IFLAG = 3, 4, 5, 6,   C
C               OR 7, RESPECTIVELY, EXCEPT THAT ADAM WAS CALLED  C
C               WITH IFLAG NEGATIVE.                             C
C                                                                C
C****************************************************************C
C                                                                C
C  THIS SUBROUTINE IS WRITTEN IN SINGLE PRECISION.               C
C  WITH MOST COMPILERS, A DOUBLE PRECISION VERSION, FOR USE WITH C
C  DOUBLE PRECISION VERSIONS OF SUBROUTINES STEP AND SINTRP, MAY C
C  BE OBTAINED SIMPLY BY SETTING THE APPROPRIATE VALUE FOR FOURU C
C  IN THE DATA STATEMENT BELOW AND BY REMOVING THE C FROM        C
C  COLUMN 1 OF THE NEXT SEVEN CARDS.                             C
C     DOUBLE PRECISION ABS,ABSDT,ABSEPS,ABSERR,AMAX1,DABS,DMAX1,
C    1                 DSIGN,DT,DTSGN,D1,D2,EPS,FOURU,H,HOLD,
C    2                 PHI,RELEPS,RELERR,RVAR,SIGN,T,TEND,TOUT,
C    3                 WT,X,Y,YP,YY
```

```
C     ABS(D1) = DABS(D1)
C     SIGN(D1,D2) = DSIGN(D1,D2)
C     AMAX1(D1,D2) = DMAX1(D1,D2)
C                                                                      C
      LOGICAL NORND,PHASE1,START
C                                                                      C
      DIMENSION Y(NEQN),YP(NEQN),YY(NEQN),WT(NEQN),PHI(NEQN,16)
C                                                                      C
      COMMON /ADDATA/ T,TOUT,RELERR,ABSERR,DTSGN,NEQN,IFLAG,
     1                IOLD,KLE4,INIT,KOP
C                                                                      C
      COMMON /STDATA/ X,H,HOLD,EPS,RVAR(86),NEQ,KOLD,ICRASH,K,
     1                NFE,NS,NORND,PHASE1,START
C                                                                      C
      EXTERNAL F
C                                                                      C
C                                                                      C
C  IN THE FOLLOWING DATA STATEMENT, FOURU MUST BE SET TO A             C
C  VALUE APPROPRIATE TO THE MACHINE BEING USED.                        C
C                                                                      C
C  FOURU IS 4.*U, WHERE THE COMPUTER UNIT ROUNDOFF ERROR U IS          C
C  THE SMALLEST POSITIVE VALUE REPRESENTABLE IN THE MACHINE            C
C  SUCH THAT (1.+U).GT.1.                                              C
C                                                                      C
C  SOME REPRESENTATIVE VALUES ARE                                      C
C    U = 9.54E-7        FOURU = 3.82E-6        IBM 360/370 S.P.        C
C    U = 5.96E-8        FOURU = 2.39E-7        PDP-11 S.P.             C
C    U = 1.49E-8        FOURU = 5.97E-8        UNIVAC 1108             C
C    U = 7.45E-9        FOURU = 2.99E-8        PDP-10                  C
C    U = 7.11E-15       FOURU = 2.85E-14       CDC 6000 SERIES         C
C    U = 2.22D-16       FOURU = 8.89D-16       IBM 360/370 D.P.        C
C    U = 1.39D-17       FOURU = 5.56D-17       PDP-11 D.P.             C
C                                                                      C
      DATA FOURU/2.39E-7/
C                                                                      C
C                                                                      C
C  THE EXPENSE IS CONTROLLED BY RESTRICTING THE NUMBER                 C
C  OF FUNCTION EVALUATIONS TO BE APPROXIMATELY MAXNFE.                 C
C  AS SET, THIS CORRESPONDS TO ABOUT 1500 STEPS.                       C
C                                                                      C
      DATA MAXNFE/3000/
C                                                                      C
C**********************************************************************C
C                                                                      C
C  TEST FOR IMPROPER PARAMETERS                                        C
C                                                                      C
      MFLAG=IABS(IFLAG)
      IF ((MFLAG.GT.7).OR.(MFLAG.EQ.0).OR.(NEQN.LT.1)) GO TO 200
      IF ((RELERR.LT.0.0).OR.(ABSERR.LT.0.0)) GO TO 200
      EPS=AMAX1(RELERR,ABSERR)
      IF (EPS.LE.0.0) GO TO 200
C                                                                      C
```

```
      NEQ=NEQN
      DT=TOUT-T
      ABSDT=ABS(DT)
      TEND=T + 10.*DT
      IF(IFLAG.LT.0) TEND=TOUT
      RELEPS=RELERR/EPS
      ABSEPS=ABSERR/EPS
C                                                                C
C  IF THIS IS THE FIRST CALL, GO TO INITIALIZATION SECTION.      C
C                                                                C
      IF (MFLAG.EQ.1) GO TO 150
C                                                                C
C  SINCE THIS IS NOT THE FIRST CALL, T=TOUT IS NOT VALID         C
C  INPUT.                                                        C
C                                                                C
      IF (T.EQ.TOUT) GO TO 200
C                                                                C
C  IF INITIALIZATION WAS NOT COMPLETED ON THE FIRST CALL, DO     C
C  IT NOW.                                                       C
C                                                                C
      IF (INIT.EQ.0) GO TO 180
C                                                                C
C  IF THE LAST STEP WAS MADE WITH IFLAG NEGATIVE, OR IF THE      C
C  DIRECTION OF INTEGRATION IS CHANGED, RESTART THE INTEGRATION. C
C                                                                C
      IF (IOLD .LT. 0) GO TO 160
      IF (DTSGN*DT .LT. 0.0) GO TO 160
C                                                                C
C  IF ALREADY PAST OUTPUT POINT, INTERPOLATE AND RETURN.         C
C                                                                C
   50 IF(ABS(X-T) .LT. ABSDT)GO TO 60
      CALL SINTRP(TOUT,Y,YP,YY,PHI)
      IOLD=IFLAG
      IFLAG=2
      T=TOUT
      RETURN
C                                                                C
C  IF CANNOT GO PAST OUTPUT POINT, TEST WHETHER TOO CLOSE TO     C
C  INTEGRATE WITH STEP.                                          C
C                                                                C
   60 IF ((IFLAG.LT.0).AND.(ABS(TOUT-X).LT.FOURU*ABS(X))) GO TO 210
C                                                                C
C  TEST FOR TOO MUCH WORK.                                       C
C                                                                C
      IF(NFE .GT. MAXNFE) GO TO 230
C                                                                C
C  LIMIT STEPSIZE, SET WEIGHT VECTOR, AND TAKE A STEP.           C
C                                                                C
      IF (ABS(H).LT.ABS(TEND-X)) GO TO 100
      KOP=KOP+1
      IF (KOP.GT.50) GO TO 290
      H=TEND-X
```

```
C                                                                C
  100 DO 110 L=1,NEQN
        WT(L)=RELEPS*ABS(YY(L))+ABSEPS
        IF (WT(L).EQ.0.0) GO TO 270
  100   CONTINUE
C                                                                C
      CALL STEP(F,YY,YP,WT,PHI,Y)
C                                                                C
C  TEST WHETHER STEP WAS SUCCESSFUL.                             C
C                                                                C
      IF (ICRASH.NE.0) GO TO 250
C                                                                C
C  STEP WAS SUCCESSFUL.  TEST FOR STIFFNESS.                     C
C                                                                C
      IF (KLE4.GE.50) GO TO 50
      KLE4=KLE4+1
      IF(KOLD .GT. 4) KLE4=0
      GO TO 50
C                                                                C
C****************************************************************C
C                                                                C
C  INITIALIZATION SECTION --                                     C
C                                                                C
C                                                                C
C  ON START AND RESTART, SET WORK VARIABLES X AND YY(*), STORE   C
C  THE DIRECTION OF INTEGRATION, AND INITIALIZE THE STEPSIZE.    C
C                                                                C
  150 NFE=0
  160 START=.TRUE.
      KLE4=0
      INIT=0
      KOP=0
      X=T
      DO 170 L=1,NEQN
  170   YY(L)=Y(L)
C                                                                C
C  IF T=TOUT ON FIRST CALL, EVALUATE DERIVATIVES AND RETURN.     C
C                                                                C
      IF (T.NE.TOUT) GO TO 180
      CALL F(X,YY,YP)
      NFE=NFE+1
      IFLAG=2
      RETURN
C                                                                C
C  SET STEPSIZE AND TRANSFER TO BEGIN INTEGRATION.               C
C                                                                C
  180 INIT=1
      DTSGN=DT
      H=SIGN(AMAX1(ABS(TOUT-X),FOURU*ABS(X)),TOUT-X)
      GO TO 50
C                                                                C
C****************************************************************C
```

```
C                                                                 C
C  THE REMAINDER OF THE PROGRAM CONTAINS STATEMENTS TO HANDLE     C
C  VARIOUS EXCEPTIONAL OCCURRENCES.                               C
C                                                                 C
C                                                                 C
C  INVALID INPUT WAS PASSED TO ADAM.                              C
C                                                                 C
  200 IFLAG=9
      RETURN
C                                                                 C
C  CANNOT PASS TOUT, BUT TOO CLOSE TO CALL STEP.  EXTRAPOLATE     C
C  BY EULER METHOD AND RETURN.                                    C
C                                                                 C
  210 H=TOUT-X
      CALL F(X,YY,YP)
      NFE=NFE+1
      DO 220 L=1,NEQN
  220   Y(L)=YY(L)+H*YP(L)
      IFLAG=2
      T=TOUT
      CALL F(T,Y,YP)
      NFE=NFE+1
      IOLD=-1
      RETURN
C                                                                 C
C  NUMBER OF FUNCTION CALLS EXCEEDED MAXNFE.  SET IFLAG TO        C
C  INDICATE WHETHER THIS RESULTS FROM STIFFNESS, AND RESET        C
C  COUNTER NFE.                                                   C
C                                                                 C
  230 IFLAG=ISIGN(5,IFLAG)
      IF(KLE4.GE.50) IFLAG=ISIGN(6,IFLAG)
      NFE=NFE-MAXNFE
      KLE4=0
      GO TO 300
C                                                                 C
C  ERROR TOLERANCES WERE TOO SMALL TO PERMIT A SUCCESSFUL STEP.   C
C  INCREASE RELERR AND ABSERR.                                    C
C                                                                 C
  250 IFLAG=ISIGN(4,IFLAG)
      IF (ICRASH.EQ.2) IFLAG=ISIGN(3,IFLAG)
      RELERR=EPS*RELEPS
      ABSERR=EPS*ABSEPS
      GO TO 300
C                                                                 C
C  A COMPONENT OF THE SOLUTION VANISHED WHEN A PURE RELATIVE      C
C  ERROR TEST WAS SPECIFIED.  USER MUST SET ABSERR POSITIVE       C
C  IN ORDER TO PROCEED.                                           C
C                                                                 C
  270 IFLAG=8
      GO TO 300
C                                                                 C
C  THE OUTPUT INTERVAL HAS BEEN TOO SMALL TO ALLOW EFFICIENT      C
```

```
C  OPERATION OF ADAM.                                                  C
C                                                                      C
  290 KOP=0
      IFLAG=ISIGN(7,IFLAG)
C                                                                      C
C  FOR ERROR RETURN, SET T, Y(*), AND YP(*) TO THE MOST RECENT         C
C  VALUES REACHED IN THE INTEGRATION.                                  C
C                                                                      C
  300 T=X
      IOLD=1
      DO 310 L=1,NEQN
        Y(L)=YY(L)
  310   YP(L)=PHI(L,1)
      RETURN
C                                                                      C
C**********************************************************************C
C                                                                      C
C  END OF ADAM                                                         C
C                                                                      C
      END


      SUBROUTINE STEP(F,Y,YP,WT,PHI,P)
C                                                                      C
C**********************************************************************C
C                                                                      C
C  SUBROUTINE STEP USES A MODIFIED DIVIDED DIFFERENCE FORM OF          C
C  AN ADAMS PECE METHOD (OF ORDER.LE.12) TO PERFORM ONE STEP           C
C  IN THE INTEGRATION OF A SYSTEM OF NEQN FIRST ORDER ORDINARY         C
C  DIFFERENTIAL EQUATIONS                                              C
C                                                                      C
C    DY/DX = F(X,Y) ,  Y = (Y(1),Y(2),...,Y(NEQN))                     C
C                                                                      C
C  LOCAL EXTRAPOLATION IS USED TO IMPROVE STABILITY AND ACCURACY.C
C                                                                      C
C  TO SPECIFY THE DIFFERENTIAL EQUATIONS TO BE SOLVED, THE USER        C
C  MUST SUPPLY A SUBROUTINE, DENOTED HERE BY F, TO                     C
C  CALCULATE THE NEQN DERIVATIVES DY(I)/DX.                            C
C                                                                      C
C    F MUST BE OF THE FORM F(X,Y,YP), WHERE Y AND YP ARE               C
C    ARRAYS EACH OF DIMENSION NEQN.  F MUST CALCULATE THE              C
C    VALUES OF THE DERIVATIVES DY(I)/DX AT (X,Y) AND PLACE             C
C    THEM IN YP.  X AND Y MUST NOT BE ALTERED BY F.                    C
C                                                                      C
C  INFORMATION IS EXCHANGED WITH STEP BOTH THROUGH THE ARGUMENT        C
C  LIST AND THROUGH THE COMMON BLOCK STDATA.                           C
C                                                                      C
C  THE PARAMETERS IN THE ARGUMENT LIST ARE                             C
C                                                                      C
C    F -- THE NAME OF THE USER-SUPPLIED SUBROUTINE TO                  C
C       CALCULATE THE DERIVATIVE                                       C
C    Y -- AN ARRAY OF DIMENSION NEQN WHICH CONTAINS THE                C
```

```
C       SOLUTION.  ITS VALUE MUST NOT BE ALTERED BETWEEN               C
C       CALLS TO STEP.                                                 C
C    YP -- AN ARRAY OF DIMENSION NEQN WHICH CONTAINS THE               C
C       DERIVATIVE OF THE SOLUTION AFTER A SUCCESSFUL STEP.            C
C       ITS VALUE NEED NOT BE PRESERVED BETWEEN STEPS.                 C
C    WT -- AN ARRAY OF DIMENSION NEQN USED IN SPECIFYING THE           C
C       ERROR TEST.                                                    C
C    PHI -- AN ARRAY OF DIMENSIONS (NEQN,16) WHICH CONTAINS            C
C       THE MODIFIED DIVIDED DIFFERENCES.  PHI(*,15) AND               C
C       PHI(*,16) ARE USED FOR PROPAGATED ROUNDOFF CONTROL.            C
C       PHI MUST NOT BE CHANGED BETWEEN CALLS.                         C
C    P -- AN ARRAY OF DIMENSION NEQN USED FOR WORKING                  C
C       STORAGE.  ITS VALUE IS NOT NEEDED FOR SUBSEQUENT               C
C       CALLS.                                                         C
C                                                                      C
C  THE PARAMETERS IN THE COMMON BLOCK STDATA ARE                       C
C                                                                      C
C    X -- THE INDEPENDENT VARIABLE                                     C
C    H -- THE STEPSIZE TO BE USED IN ATTEMPTING THE NEXT STEP          C
C    HOLD -- THE STEPSIZE USED FOR THE LAST SUCCESSFUL STEP            C
C    EPS -- THE DESIRED TOLERANCE FOR THE ERROR TEST                   C
C    NEQN -- THE NUMBER OF EQUATIONS BEING INTEGRATED                  C
C    KOLD -- THE ORDER OF THE METHOD USED FOR THE LAST                 C
C       SUCCESSFUL STEP                                                C
C    ICRASH -- AN INDICATOR OF ERROR CONDITIONS UPON RETURN            C
C    K -- THE ORDER OF THE METHOD TO BE USED FOR THE NEXT STEP         C
C    NFE -- A COUNTER WHICH IS INCREMENTED BY 1 AFTER EACH             C
C       CALL TO F                                                      C
C    START -- A LOGICAL VARIABLE WHICH MUST BE SET .TRUE. ON           C
C       THE FIRST CALL TO STEP                                         C
C    ALPHA(12),BETA(12),G(13),SIG(13),V(12),W(12),                     C
C    PSI(12),NS,NORND,PHASE1 -- QUANTITIES USED FOR INTERNAL           C
C       BOOKKEEPING BY STEP WHOSE VALUES ARE NEEDED FOR                C
C       CONTINUING INTEGRATION.  THEIR VALUES ARE NORMALLY OF          C
C       NO INTEREST TO THE USER.  THEY ARE PLACED IN THE               C
C       COMMON BLOCK TO AVOID LOCAL RETENTION BETWEEN CALLS.           C
C       NORND AND PHASE1 ARE LOGICAL VARIABLES.                        C
C                                                                      C
C  EXCEPT FOR H, EPS, AND NFE, THE VALUES IN THE COMMON BLOCK          C
C  MUST NOT BE ALTERED BETWEEN CALLS TO STEP.                          C
C                                                                      C
C  THE COUNTER NFE IS PROVIDED SO THAT THE USER CAN MONITOR THE        C
C  AMOUNT OF WORK BEING DONE.  ITS VALUE HAS NO MEANING TO STEP.       C
C                                                                      C
C----------------------------------------------------------------------C
C                                                                      C
C  STEP AUTOMATICALLY ADJUSTS THE STEPSIZE AND THE ORDER OF THE        C
C  METHOD SO THAT THE MAX NORM OF THE VECTOR WITH COMPONENTS           C
C  LOCAL ERROR(I)/WT(I) IS LESS THAN EPS FOR EACH STEP.  THE           C
C  ARRAY WT ALLOWS THE USER TO SPECIFY AN ERROR TEST WHICH IS          C
C  APPROPRIATE FOR HIS PROBLEM.                                        C
C                                                                      C
```

```
C  FOR EXAMPLE,                                                   C
C                                                                 C
C    WT(I) = 1.0  SPECIFIES ABSOLUTE ERROR                        C
C                                                                 C
C    WT(I) = ABS(Y(I)) SPECIFIES ERROR RELATIVE TO THE MOST       C
C                 RECENT VALUE OF THE I-TH COMPONENT OF THE       C
C                 SOLUTION                                        C
C                                                                 C
C    WT(I) = AMAX1(WT(I),ABS(Y(I))) SPECIFIES ERROR RELATIVE      C
C                 TO THE LARGEST MAGNITUDE YET ACHIEVED BY THE    C
C                 I-TH COMPONENT OF THE SOLUTION                  C
C                                                                 C
C    WT(I) = ABS(Y(I))*RELERR/EPS+ABSERR/EPS, WHERE EPS =         C
C                 AMAX1(RELERR,ABSERR), SPECIFIES A MIXED ERROR   C
C                 TEST WHERE RELERR IS RELATIVE ERROR AND         C
C                 ABSERR IS ABSOLUTE ERROR                        C
C                                                                 C
C  ALL COMPONENTS OF WT MUST BE NONZERO.                          C
C                                                                 C
C-----------------------------------------------------------------C
C                                                                 C
C  UPON RETURN, THE VALUE OF ICRASH INDICATES THE ACTION WHICH    C
C  WAS TAKEN.                                                     C
C                                                                 C
C    ICRASH = 0 -- A STEP OF LENGTH HOLD WAS SUCCESSFULLY         C
C       TAKEN TO THE VALUE NOW IN X.  THE SOLUTION AT X           C
C       IS IN Y, AND YP CONTAINS THE DERIVATIVES.  THE            C
C       DERIVATIVES ARE ALSO IN PHI(*,1).                         C
C                                                                 C
C    ICRASH = 1 -- NO STEP WAS TAKEN BECAUSE H WAS SMALLER THAN   C
C       PERMITTED BY THE MACHINE PRECISION.  H WAS INCREASED      C
C       TO AN ACCEPTABLE VALUE.                                   C
C                                                                 C
C    ICRASH = 2 -- NO STEP WAS TAKEN BECAUSE EPS WAS TOO SMALL    C
C       FOR THE MACHINE PRECISION.  EPS WAS INCREASED TO AN       C
C       ACCEPTABLE VALUE.                                         C
C                                                                 C
C    ICRASH = 3 -- NO STEP WAS TAKEN BECAUSE THE ERROR TEST       C
C       COULD NOT BE SATISFIED AT THE SMALLEST ALLOWED STEPSIZE.  C
C       EPS IS INCREASED TO A VALUE WHICH IS ACHIEVABLE.          C
C                                                                 C
C-----------------------------------------------------------------C
C                                                                 C
C  FIRST CALL TO STEP                                             C
C  ------------------                                             C
C                                                                 C
C  TO BEGIN THE SOLUTION, THE CALLING PROGRAM MUST                C
C                                                                 C
C    1. PROVIDE STORAGE FOR THE ARRAYS Y(NEQN), YP(NEQN),         C
C       WT(NEQN), PHI(NEQN,16), AND P(NEQN).                      C
C                                                                 C
C    2. PLACE THE INITIAL VALUES OF THE SOLUTION IN Y.            C
```

```
C                                                                C
C    3. SET WT(1), ..., WT(NEQN) TO NONZERO VALUES TO PROVIDE    C
C       THE ERROR TEST DESIRED.                                  C
C                                                                C
C    4. INITIALIZE THE FOLLOWING VARIABLES IN THE COMMON BLOCK   C
C       STDATA                                                   C
C          X -- THE STARTING VALUE OF THE INDEPENDENT VARIABLE   C
C          H -- A NOMINAL STEPSIZE FOR THE FIRST STEP,           C
C             INDICATING THE DIRECTION OF INTEGRATION AND THE    C
C             MAXIMUM STEPSIZE.  H WILL BE REDUCED AS NECESSARY  C
C             TO MEET THE ERROR CRITERION.                       C
C          EPS -- THE DESIRED ERROR TOLERANCE                    C
C          NEQN -- THE NUMBER OF EQUATIONS                       C
C          NFE -- SET TO 0 TO BEGIN A COUNT OF FUNCTION CALLS    C
C          START -- SET .TRUE.                                   C
C                                                                C
C    5. PLACE THE ACTUAL NAME OF THE USER SUPPLIED SUBROUTINE    C
C       F(X,Y,YP) IN THE CALL LIST, AND DECLARE THAT NAME        C
C       IN AN EXTERNAL STATEMENT.                                C
C                                                                C
C                                                                C
C  SUBSEQUENT CALLS TO STEP                                      C
C  ------------------------                                      C
C                                                                C
C  STEP RETURNS WITH ALL INFORMATION NEEDED TO CONTINUE THE      C
C  INTEGRATION.  TO MAINTAIN RELATIVE ERROR TESTS AS DESCRIBED   C
C  ABOVE, WT(*) MUST BE UPDATED AFTER EACH STEP.  CALLING STEP   C
C  AGAIN THEN ADVANCES THE SOLUTION ANOTHER STEP.  TO CONTINUE   C
C  INTEGRATION USING THE REVISED STEPSIZE OR TOLERANCE           C
C  PROVIDED WITH AN ERROR RETURN (ICRASH.NE.0), IT IS ONLY       C
C  NECESSARY TO CALL STEP AGAIN.                                 C
C                                                                C
C  NORMALLY, THE INTEGRATION IS CONTINUED TO THE FIRST STEP      C
C  BEYOND THE DESIRED OUTPUT POINT, AND THE SOLUTION IS INTER-   C
C  POLATED TO THE OUTPUT POINT USING THE SUBROUTINE SINTRP.      C
C  IF IT IS IMPOSSIBLE TO INTEGRATE BEYOND THE OUTPUT POINT, H   C
C  MAY BE REDUCED TO HIT THE OUTPUT POINT.  OTHERWISE, H SHOULD  C
C  NOT BE ALTERED, SINCE CHANGING H IMPAIRS THE EFFICIENCY OF    C
C  THE CODE.  H MUST NEVER BE INCREASED.  EPS MAY BE CHANGED     C
C  BETWEEN CALLS, BUT LARGE DECREASES SHOULD BE AVOIDED, SINCE   C
C  THIS WILL INVALIDATE THE STEP SELECTION MECHANISM AND LEAD    C
C  TO STEP FAILURES.                                             C
C                                                                C
C****************************************************************C
C                                                                C
C  THIS SUBROUTINE IS WRITTEN IN SINGLE PRECISION.               C
C  WITH MOST COMPILERS, A DOUBLE PRECISION VERSION MAY BE        C
C  OBTAINED BY SETTING THE APPROPRIATE VALUE FOR FOURU IN THE    C
C  DATA STATEMENT BELOW, AND BY REMOVING THE C FROM COLUMN 1 OF  C
C  THE NEXT ELEVEN CARDS.                                        C
C     DOUBLE PRECISION ABS,ABSH,ALPHA,AMAX1,AMIN1,BETA,DABS,
C    1                 DMAX1,DMIN1,DSIGN,DSQRT,DUM1,DUM2,EPS,
C     2                  ERK,ERKM1,ERKM2,ERKP1,ERR,FOURU,G,GSTR,
C     3                  H,HNEW,HOLD,P,PHI,PSI,R,REALNS,RHO,ROUND,
C     4                  SIG,SIGN,SQRT,SUM,TAU,TEMP1,TEMP2,TEMP3,
C     5                  TEMP4,TEMP5,TEMP6,TWO,V,W,WT,X,XOLD,Y,YP
C     ABS(DUM1)=DABS(DUM1)
C     AMAX1(DUM1,DUM2)=DMAX1(DUM1,DUM2)
C     AMIN1(DUM1,DUM2)=DMIN1(DUM1,DUM2)
C     SIGN(DUM1,DUM2)=DSIGN(DUM1,DUM2)
C     SQRT(DUM1)=DSQRT(DUM1)
C                                                                      C
      LOGICAL START,PHASE1,NORND
C                                                                      C
      EXTERNAL F
C                                                                      C
      COMMON /STDATA/ X,H,HOLD,EPS,ALPHA(12),BETA(12),G(13),
     1                 SIG(13),V(12),W(12),PSI(12),NEQN,KOLD,
     2                 ICRASH,K,NFE,NS,NORND,PHASE1,START
C                                                                      C
      DIMENSION Y(NEQN),WT(NEQN),PHI(NEQN,16),P(NEQN),
     1          YP(NEQN),GSTR(13),TWO(12)
C                                                                      C
      DATA TWO/8.,16.,32.,64.,128.,256.,512.,1024.,2048.,4096.,
     1          8192.,16384./
C                                                                      C
      DATA GSTR/.5,.0833,.0417,.0264,.0188,.0143,.0114,
     1          .00936,.00789,.00697,.00592,.00524,.00468/
C                                                                      C
C                                                                      C
C  IN THE FOLLOWING DATA STATEMENT, FOURU MUST BE SET TO A             C
C  VALUE APPROPRIATE TO THE MACHINE BEING USED.                        C
C                                                                      C
C  FOURU IS 4.*U, WHERE THE COMPUTER UNIT ROUNDOFF ERROR U IS          C
C  THE SMALLEST POSITIVE VALUE REPRESENTABLE IN THE MACHINE            C
C  SUCH THAT (1.+U).GT.1.                                              C
C                                                                      C
C  SOME REPRESENTATIVE VALUES ARE                                      C
C    U = 9.54E-7        FOURU = 3.82E-6        IBM 360/370 S.P.        C
C    U = 5.96E-8        FOURU = 2.39E-7        PDP-11 S.P.             C
C    U = 1.49E-8        FOURU = 5.97E-8        UNIVAC 1108             C
C    U = 7.45E-9        FOURU = 2.99E-8        PDP-10                  C
C    U = 7.11E-15       FOURU = 2.85E-14       CDC 6000 SERIES         C
C    U = 2.22D-16       FOURU = 8.89D-16       IBM 360/370 D.P.        C
C    U = 1.39D-17       FOURU = 5.56D-17       PDP-11 D.P.             C
C                                                                      C
      DATA FOURU/2.39E-7/
C                                                                      C
C**********************************************************************C
C                  ******                                              C
C  BEGIN BLOCK 0   ******                                              C
C                  ******                                              C
C************************                                              C
C                                                                      C
```

```
C  CHECK WHETHER STEPSIZE OR ERROR TOLERANCE IS TOO SMALL FOR    C
C  MACHINE PRECISION.  IF FIRST STEP, INITIALIZE PHI ARRAY AND   C
C  ESTIMATE A STARTING STEPSIZE.                                 C
C                                                                C
C                                                                C
C  IF STEPSIZE IS TOO SMALL, DETERMINE AN ACCEPTABLE ONE.        C
C                                                                C
      IF (ABS(H) .GE. FOURU*ABS(X)) GO TO 5
      H=SIGN(FOURU*ABS(X),H)
      ICRASH=1
      RETURN
C                                                                C
C  IF ERROR TOLERANCE IS TOO SMALL, INCREASE IT TO AN ACCEPTABLE C
C  VALUE.                                                        C
C                                                                C
    5 ROUND=0.
      DO 10 L=1,NEQN
   10   ROUND=AMAX1(ROUND,ABS(Y(L)/WT(L)))
      ROUND=FOURU*ROUND
      IF (EPS .GE. ROUND) GO TO 15
      EPS=ROUND*(1.+FOURU)
      ICRASH=2
      RETURN
C                                                                C
   15 ICRASH=0
      G(1)=1.0
      G(2)=0.5
      SIG(1)=1.0
      IFAIL=0
      IF (.NOT. START) GO TO 100
C                                                                C
C  INITIALIZE.  COMPUTE APPROPRIATE STEPSIZE FOR FIRST STEP.     C
C                                                                C
      CALL F(X,Y,YP)
      NFE=NFE+1
      SUM=0.
        DO 20 L=1,NEQN
        PHI(L,1)=YP(L)
        PHI(L,2)=0.
   20   SUM=AMAX1(SUM,ABS(YP(L)/WT(L)))
C                                                                C
      ABSH=ABS(H)
      IF (EPS .LT. 16.*SUM*H*H) ABSH=0.25*SQRT(EPS/SUM)
      H=SIGN(AMAX1(ABSH,FOURU*ABS(X)),H)
      HOLD=0.
      K=1
      KOLD=0
      START=.FALSE.
      PHASE1=.TRUE.
      NORND=.TRUE.
      IF (EPS .GT. 200.*ROUND) GO TO 100
      NORND=.FALSE.
      DO 25 L=1,NEQN
   25   PHI(L,15)=0.
C                                                                      C
C***********************                                               C
C                 ******                                               C
C  END BLOCK 0    ******                                               C
C                 ******                                               C
C**********************************************************************C
C                 ******                                               C
C  BEGIN BLOCK 1  ******                                               C
C                 ******                                               C
C***********************                                               C
C                                                                      C
C  COMPUTE COEFFICIENTS OF FORMULAS FOR THIS STEP.  AVOID              C
C  COMPUTING THOSE QUANTITIES NOT CHANGED WHEN STEPSIZE IS NOT         C
C  CHANGED.                                                            C
C                                                                      C
  100 KP1=K+1
      KP2=K+2
      KM1=K-1
      KM2=K-2
C                                                                      C
C  NS IS THE NUMBER OF STEPS TAKEN WITH SIZE H, INCLUDING THE          C
C  CURRENT ONE.  WHEN K.LT.NS, NO COEFFICIENTS CHANGE.                 C
C                                                                      C
      IF (H .NE. HOLD) NS=0
      NS=MIN0(NS+1,KOLD+1)
      NSP1=NS+1
      IF (K .LT. NS) GO TO 200
C                                                                      C
C  COMPUTE THOSE COMPONENTS OF ALPHA(*), BETA(*), PSI(*), AND          C
C  SIG(*) WHICH ARE CHANGED.                                           C
C                                                                      C
      BETA(NS)=1.
      REALNS=NS
      ALPHA(NS)=1./REALNS
      TEMP1=H*REALNS
      SIG(NSP1)=1.
      IF (K .LT. NSP1) GO TO 110
      DO 105 I=NSP1,K
         IM1=I-1
         TEMP2=PSI(IM1)
         PSI(IM1)=TEMP1
         BETA(I)=BETA(IM1)*PSI(IM1)/TEMP2
         TEMP1=TEMP2+H
         ALPHA(I)=H/TEMP1
  105    SIG(I+1)=FLOAT(I)*ALPHA(I)*SIG(I)
  110 PSI(K)=TEMP1
C                                                                      C
C  COMPUTE COEFFICIENTS G(*)                                           C
C                                                                      C
C  INITIALIZE V(*) AND SET W(*).                                       C
```

```
C                                                                C
      IF (NS .GT. 1) GO TO 120
      DO 115 IQ=1,K
         TEMP3=IQ*(IQ+1)
         V(IQ)=1./TEMP3
  115    W(IQ)=V(IQ)
      GO TO 140
C                                                                C
C  IF ORDER WAS RAISED, UPDATE DIAGONAL PART OF B(*).            C
C                                                                C
  120 IF (K .LE. KOLD) GO TO 130
      TEMP4=K*KP1
      V(K)=1./TEMP4
      NSM2=NS-2
      IF (NSM2 .LT. 1) GO TO 130
      DO 125 J=1,NSM2
         I=K-J
  125    V(I)=V(I)-ALPHA(J+1)*V(I+1)
C                                                                C
C  UPDATE V(*) AND SET W(*).                                     C
C                                                                C
  130 LIMIT1=KP1-NS
      TEMP5=ALPHA(NS)
      DO 135 IQ=1,LIMIT1
         V(IQ)=V(IQ)-TEMP5*V(IQ+1)
  135    W(IQ)=V(IQ)
      G(NSP1)=W(1)
C                                                                C
C  COMPUTE THE G(*) IN THE WORK VECTOR W(*).                     C
C                                                                C
  140 NSP2=NS+2
         IF (KP1 .LT. NSP2) GO TO 200
         DO 150 I=NSP2,KP1
           LIMIT2=KP2-I
           TEMP6=ALPHA(I-1)
           DO 145 IQ=1,LIMIT2
  145         W(IQ)=W(IQ)-TEMP6*W(IQ+1)
  150    G(I)=W(1)
C                                                                C
C***********************                                         C
C                 ******                                         C
C  END BLOCK 1    ******                                         C
C                 ******                                         C
C****************************************************************C
C                 ******                                         C
C  BEGIN BLOCK 2  ******                                         C
C                 ******                                         C
C***********************                                         C
C                                                                C
C  PREDICT A SOLUTION P(*), EVALUATE DERIVATIVES USING PREDICTED C
C  SOLUTION, AND ESTIMATE LOCAL ERROR AT ORDER K AND ERRORS AT   C
C  ORDERS K, K-1, K-2 AS IF CONSTANT STEPSIZE WERE USED.         C
```

```
C                                                                       C
C                                                                       C
C  CHANGE PHI TO PHI STAR.                                              C
C                                                                       C
  200 IF (K .LT. NSP1) GO TO 215
      DO 210 I=NSP1,K
         TEMP1=BETA(I)
         DO 205 L=1,NEQN
  205       PHI(L,I)=TEMP1*PHI(L,I)
  210 CONTINUE
C                                                                       C
C  PREDICT SOLUTION AND DIFFERENCES.                                    C
C                                                                       C
  215 DO 220 L=1,NEQN
         PHI(L,KP2)=PHI(L,KP1)
         PHI(L,KP1)=0.
  220    P(L)=0.
      DO 230 J=1,K
         I=KP1-J
         IP1=I+1
         TEMP2=G(I)
         DO 225 L=1,NEQN
           P(L)=P(L)+TEMP2*PHI(L,I)
  225      PHI(L,I)=PHI(L,I)+PHI(L,IP1)
  230 CONTINUE
      IF (NORND) GO TO 240
      DO 235 L=1,NEQN
         TAU=H*P(L)-PHI(L,15)
         P(L)=Y(L)+TAU
  235    PHI(L,16)=(P(L)-Y(L))-TAU
      GO TO 250
  240 DO 245 L=1,NEQN
  245   P(L)=Y(L)+H*P(L)
  250 XOLD=X
      X=X+H
      ABSH=ABS(H)
      CALL F(X,P,YP)
      NFE=NFE+1
C                                                                       C
C  ESTIMATE ERRORS AT ORDERS K, K-1, K-2.                               C
C                                                                       C
      ERKM2=0.
      ERKM1=0.
      ERK=0.
      DO 265 L=1,NEQN
         TEMP3=1./WT(L)
         TEMP4=YP(L)-PHI(L,1)
         IF (KM2) 265,260,255
  255    ERKM2=AMAX1(ERKM2,ABS((PHI(L,KM1)+TEMP4)*TEMP3))
  260    ERKM1=AMAX1(ERKM1,ABS((PHI(L,K)+TEMP4)*TEMP3))
  265    ERK=AMAX1(ERK,ABS(TEMP4*TEMP3))
      IF (KM2) 280,275,270
  270 ERKM2=ABSH*SIG(KM1)*GSTR(KM2)*ERKM2
  275 ERKM1=ABSH*SIG(K)*GSTR(KM1)*ERKM1
  280 TEMP5=ABSH*ERK
      ERR=TEMP5*(G(K)-G(KP1))
      ERK=TEMP5*SIG(KP1)*GSTR(K)
      KNEW=K
C                                                                C
C  TEST WHETHER ORDER SHOULD BE LOWERED.                         C
C                                                                C
      IF (KM2) 299,290,285
  285 IF (AMAX1(ERKM1,ERKM2) .LE. ERK) KNEW=KM1
      GO TO 299
  290 IF (ERKM1 .LE. 0.5*ERK) KNEW=KM1
C                                                                C
C  TEST WHETHER STEP WAS SUCCESSFUL.                             C
C                                                                C
  299 IF (ERR .LE. EPS) GO TO 400
C                                                                C
C***********************                                         C
C                 ******                                         C
C  END BLOCK 2    ******                                         C
C                 ******                                         C
C****************************************************************C
C                 ******                                         C
C  BEGIN BLOCK 3  ******                                         C
C                 ******                                         C
C***********************                                         C
C                                                                C
C  THE STEP IS UNSUCCESSFUL.  RESTORE X, PHI(*,*), PSI(*).       C
C  IF THIRD CONSECUTIVE FAILURE SET ORDER TO ONE.  IF STEP FAILS C
C  MORE THAN THREE TIMES, CONSIDER AN OPTIMAL STEPSIZE.  DOUBLE  C
C  ERROR TOLERANCE AND RETURN IF ESTIMATED STEPSIZE IS TOO SMALL C
C  FOR MACHINE PRECISION.                                        C
C                                                                C
C                                                                C
C  RESTORE X, PHI(*,*), AND PSI(*).                              C
C                                                                C
      PHASE1=.FALSE.
      X=XOLD
      DO 310 I=1,K
         TEMP1=1./BETA(I)
         IP1=I+1
         DO 305 L=1,NEQN
  305         PHI(L,I)=TEMP1*(PHI(L,I)-PHI(L,IP1))
  310    CONTINUE
      IF (K .LT. 2) GO TO 320
      DO 315 I=2,K
  315    PSI(I-1)=PSI(I)-H
C                                                                C
C  ON THIRD FAILURE, SET ORDER TO ONE.  THEREAFTER, USE OPTIMAL  C
C  STEPSIZE.                                                     C
C                                                                C
```

```
  320 IFAIL=IFAIL+1
      TEMP2=0.5
      IF (IFAIL-3) 335,330,325
  325 IF (EPS .LT. 0.5*ERK) TEMP2=SQRT(0.5*EPS/ERK)
  330 KNEW=1
  335 H=TEMP2*H
      K=KNEW
      IF (ABS(H) .GE. FOURU*ABS(X)) GO TO 100
C                                                                 C
      ICRASH=3
      H=SIGN(FOURU*ABS(X),H)
      EPS=EPS+EPS
      RETURN
C                                                                 C
C***********************                                          C
C                 ******                                          C
C  END BLOCK 3    ******                                          C
C                 ******                                          C
C*****************************************************************C
C                 ******                                          C
C  BEGIN BLOCK 4  ******                                          C
C                 ******                                          C
C***********************                                          C
C                                                                 C
C  THE STEP IS SUCCESSFUL.  CORRECT THE PREDICTED SOLUTION,       C
C  EVALUATE THE DERIVATIVES USING THE CORRECTED SOLUTION, AND    C
C  UPDATE THE DIFFERENCES,  DETERMINE BEST ORDER AND STEPSIZE    C
C  FOR THE NEXT STEP.                                             C
C                                                                 C
  400 KOLD=K
      HOLD=H
C                                                                 C
C  CORRECT AND EVALUATE.                                          C
C                                                                 C
      TEMP1=H*G(KP1)
      IF (NORND) GO TO 410
      DO 405 L=1,NEQN
         RHO=TEMP1*(YP(L)-PHI(L,1))-PHI(L,16)
         Y(L)=P(L)+RHO
  405    PHI(L,15)=(Y(L)-P(L))-RHO
      GO TO 420
  410 DO 415 L=1,NEQN
  415    Y(L)=P(L)+TEMP1*(YP(L)-PHI(L,1))
  420 CALL F(X,Y,YP)
      NFE=NFE+1
C                                                                 C
C  UPDATE DIFFERENCES FOR NEXT STEP.                              C
C                                                                 C
      DO 425 L=1,NEQN
         PHI(L,KP1)=YP(L)-PHI(L,1)
  425    PHI(L,KP2)=PHI(L,KP1)-PHI(L,KP2)
      DO 430 I=1,K
```

```
         DO 430 L=1,NEQN
  430         PHI(L,I)=PHI(L,I)+PHI(L,KP1)
C                                                                 C
C  ESTIMATE ERROR AT ORDER K+1 UNLESS,                            C
C     IN STARTING PHASE WHEN ORDER IS ALWAYS RAISED,              C
C     ALREADY DECIDED IN BLOCK 2 TO LOWER ORDER,                  C
C     STEPSIZE IS NOT CONSTANT SO ESTIMATE WOULD BE UNRELIABLE.  C
C                                                                 C
      ERKP1=0.
      IF ((KNEW.EQ.KM1).OR.(K.EQ.12)) PHASE1=.FALSE.
      IF (PHASE1) GO TO 450
      IF (KNEW .EQ. KM1) GO TO 455
      IF (KP1 .GT. NS) GO TO 460
      DO 440 L=1,NEQN
  440   ERKP1=AMAX1(ERKP1,ABS(PHI(L,KP2)/WT(L)))
      ERKP1=ABSH*GSTR(KP1)*ERKP1
C                                                                 C
C  USING ESTIMATED ERROR AT ORDER K+1, DETERMINE APPROPRIATE      C
C  ORDER FOR NEXT STEP.                                           C
C                                                                 C
      IF (K .GT. 1) GO TO 445
      IF (ERKP1 .GE. 0.5*ERK) GO TO 460
      GO TO 450
  445 IF (ERKM1 .LE. AMIN1(ERK,ERKP1)) GO TO 455
      IF ((ERKP1.GE.ERK).OR.(K.EQ.12)) GO TO 460
C                                                                 C
C  HERE ERKP1.LT.ERK.LT.AMAX1(ERKM1,ERKM2), ELSE ORDER WOULD      C
C  HAVE BEEN LOWERED IN BLOCK 2.  THUS, ORDER IS RAISED.          C
C                                                                 C
  450 K=KP1
      ERK=ERKP1
      GO TO 460
C                                                                 C
C  LOWER ORDER.                                                   C
C                                                                 C
  455 K=KM1
      ERK=ERKM1
C                                                                 C
C  WITH NEW ORDER DETERMINE APPROPRIATE STEPSIZE FOR NEXT STEP.  C
C                                                                 C
  460 HNEW=H+H
      IF (PHASE1) GO TO 465
      IF (EPS .GE. ERK*TWO(K)) GO TO 465
      HNEW=H
      IF (EPS .GE. 2.*ERK) GO TO 465
      R=(0.5*EPS/ERK)**(1./FLOAT(K+1))
      IF (R.GT.0.9) R=0.9
      IF (R.LT.0.5) R=0.5
      HNEW=SIGN(AMAX1(ABSH*R,FOURU*ABS(X)),H)
  465 H=HNEW
      RETURN
C                                                                 C
```

```
C***********************                                         C
C                 ******                                         C
C  END BLOCK 4    ******                                         C
C                 ******                                         C
C****************************************************************C
C                                                                C
C  END OF STEP                                                   C
C                                                                C
      END


      SUBROUTINE SINTRP(XOUT,YOUT,YPOUT,Y,PHI)
C                                                                C
C****************************************************************C
C                                                                C
C  SUBROUTINE SINTRP MAY BE USED IN CONJUNCTION WITH SUBROUTINE  C
C  STEP TO INTERPOLATE THE SOLUTION TO THE NONMESH POINT XOUT.   C
C  SINTRP SHOULD BE CALLED WHEN THE INTEGRATION HAS PROCEEDED TO C
C  THE FIRST MESH POINT BEYOND XOUT.                             C
C                                                                C
C  THE METHODS IN SUBROUTINE STEP APPROXIMATE THE SOLUTION NEAR  C
C  X BY A POLYNOMIAL.  SINTRP APPROXIMATES THE SOLUTION AT       C
C  XOUT BY EVALUATING THE POLYNOMIAL THERE.                      C
C                                                                C
C  THE PARAMETERS IN THE ARGUMENT LIST ARE                       C
C                                                                C
C    XOUT -- THE VALUE OF THE INDEPENDENT VARIABLE FOR WHICH     C
C            THE SOLUTION IS TO BE FOUND                         C
C    YOUT -- AN ARRAY OF DIMENSION NEQN TO HOLD THE VALUE OF     C
C            THE SOLUTION AT XOUT.                               C
C    YPOUT -- AN ARRAY OF DIMENSION NEQN TO HOLD THE VALUE OF    C
C             DERIVATIVE AT XOUT.                                C
C    Y -- THE ARRAY OF DIMENSION NEQN WHICH CONTAINS THE         C
C         SOLUTION RETURNED AT X BY STEP                         C
C    PHI -- THE ARRAY OF DIMENSIONS (NEQN,16) WHICH HOLDS THE    C
C           MODIFIED DIVIDED DIFFERENCES RETURNED AT X BY STEP   C
C                                                                C
C  THE REMAINING PARAMETERS X, PSI(12), NEQN, AND KOLD WHICH     C
C  ARE NEEDED FOR DEFINING THE POLYNOMIAL, ARE PASSED FROM       C
C  SUBROUTINE STEP THROUGH THE COMMON BLOCK STDATA.              C
C                                                                C
C****************************************************************C
C                                                                C
C  THIS PROGRAM IS WRITTEN IN SINGLE PRECISION.  TO OBTAIN A     C
C  DOUBLE PRECISION VERSION, REMOVE THE C FROM COLUMN 1 OF       C
C  THE NEXT THREE CARDS.                                         C
C     DOUBLE PRECISION XOUT,YOUT,YPOUT,Y,PHI,G,W,RHO,X,RVAR,
C    1                 PSI,HI,TEMP1,TERM,PSIJM1,GAMMA,ETA,
C    2                 TEMP2,TEMP3
C                                                                C
      LOGICAL LVAR
C                                                                C
```

```
      DIMENSION Y(NEQN),YOUT(NEQN),YPOUT(NEQN),PHI(NEQN,16),
     1           G(13),W(13),RHO(13)
C                                                                      C
      COMMON /STDATA/ X,RVAR(77),PSI(12),NEQN,KOLD,IVAR(4),LVAR(3)
C                                                                      C
      DATA G(1)/1.0/,RHO(1)/1.0/
C                                                                      C
      HI=XOUT-X
      KI=KOLD+1
      KIP1=KI+1
C                                                                      C
C   INITIALIZE W(*) FOR COMPUTING G(*).                                C
C                                                                      C
      DO 5 I=1,KI
         TEMP1=I
    5    W(I)=1.0/TEMP1
      TERM=0.0
C                                                                      C
C   COMPUTE G(*).                                                      C
C                                                                      C
      DO 15 J=2,KI
         JM1=J-1
         PSIJM1=PSI(JM1)
         GAMMA=(HI+TERM)/PSIJM1
         ETA=HI/PSIJM1
         LIMIT1=KIP1-J
         DO 10 I=1,LIMIT1
   10      W(I)=GAMMA*W(I)-ETA*W(I+1)
         G(J)=W(1)
         RHO(J)=GAMMA*RHO(JM1)
   15    TERM=PSIJM1
C                                                                      C
C   INTERPOLATE.                                                       C
C                                                                      C
      DO 20 L=1,NEQN
         YPOUT(L)=0.0
   20    YOUT(L)=0.0
      DO 25 J=1,KI
         I=KIP1-J
         TEMP2=G(I)
         TEMP3=RHO(I)
         DO 25 L=1,NEQN
           YOUT(L)=YOUT(L)+TEMP2*PHI(L,I)
   25      YPOUT(L)=YPOUT(L)+TEMP3*PHI(L,I)
      DO 35 L=1,NEQN
   35    YOUT(L)=Y(L)+HI*YOUT(L)
C                                                                      C
      RETURN
C                                                                      C
C  END OF SINTRP                                                       C
C                                                                      C
      END
```

Appendix C

Stiff Method Programs

This appendix contains the complete listings of the subroutines STFINT, STIFF, GEAR, DECOMP, GINTRP, and QJACOB.

```
      SUBROUTINE STFINT(NEQN,Y,W,IW,TINIT,TFINAL,TINCR,RELERR,
     1                  ABSERR,IUNIT,FCT,FDER,OUTP)
C                                                                C
C****************************************************************C
C                                                                C
C  SUBROUTINE STFINT USES STIFF TO INTEGRATE A SYSTEM OF NEQN    C
C  FIRST ORDER ORDINARY DIFFERENTIAL EQUATIONS                   C
C                                                                C
C    DY/DT = F(T,Y) ,  Y = (Y(1),Y(2),...,Y(NEQN))               C
C                                                                C
C  FROM TINIT TO TFINAL, WITH OUTPUT OF THE SOLUTION AT          C
C  INTERVALS OF TINCR BY MEANS OF CALLS TO THE USER-             C
C  -SUPPLIED SUBROUTINE OUTP.  STIFF IS INTENDED FOR             C
C  SOLVING STIFF PROBLEMS, AND IS NOT AN EFFICIENT MEANS OF      C
C  SOLVING NONSTIFF PROBLEMS.                                    C
C                                                                C
C                                                                C
C  THE QUANTITIES IN THE ARGUMENT LIST REPRESENT                 C
C                                                                C
C    NEQN -- THE NUMBER OF EQUATIONS                             C
C    Y -- AN ARRAY OF DIMENSION NEQN, WHICH CONTAINS THE         C
C          VALUE OF THE SOLUTION                                 C
C    W -- AN ARRAY OF DIMENSION NEQN**2+17*NEQN, WHICH IS USED   C
C          FOR WORKING STORAGE.                                  C
C    IW -- AN INTEGER ARRAY OF DIMENSION NEQN, WHICH IS USED FOR C
C           WORKING STORAGE.                                     C
C    TINIT -- THE STARTING VALUE OF THE INDEPENDENT VARIABLE     C
C    TFINAL -- THE VALUE OF THE INDEPENDENT VARIABLE AT WHICH    C
C               INTEGRATION IS TO STOP                           C
C    TINCR -- THE DESIRED SPACING, IN THE INDEPENDENT VARIABLE,  C
C              OF THE OUTPUT POINTS                              C
C    RELERR -- THE DESIRED RELATIVE ERROR TOLERANCE              C
C    ABSERR -- THE DESIRED ABSOLUTE ERROR TOLERANCE              C
C    IUNIT -- THE LOGICAL UNIT NUMBER OF THE DATA SET TO BE      C
C              USED FOR WRITING DIAGNOSTIC MESSAGES.  IF IUNIT   C
C              IS LESS THAN OR EQUAL TO ZERO, NO MESSAGES ARE    C
C              WRITTEN                                           C
```

```
C    FCT -- THE NAME OF THE USER-SUPPLIED SUBROUTINE                   C  13700
C           FCT(T,Y,YP) WHICH EVALUATES THE DERIVATIVES                C  13800
C           YP(I)=DY(I)/DT, I=1,2,...,NEQN                             C  13900
C   FDER -- THE NAME OF THE USER SUPPLIED SUBROUTINE WHICH             C  14000
C            EVALUATES THE JACOBIAN MATRIX AS DESCRIBED IN THE         C  14100
C            COMMENTS TO GEAR.                                         C  14200
C   OUTP -- THE NAME OF THE USER-SUPPLIED SUBROUTINE                   C  14300
C            OUTP(T,Y,W) USED TO HANDLE OUTPUT.                        C  14400
C                                                                      C  14500
C                                                                      C  14600
C TO USE STFINT, THE CALLING PROGRAM MUST                              C  14700
C                                                                      C  14800
C   1. ALLOCATE SPACE FOR THE ARRAYS Y(*), IW(*), AND W(*) IN A        C  14900
C      DIMENSION STATEMENT.  (NOTE THAT Y MUST BE AN ARRAY,            C  15000
C      EVEN IF NEQN=1.)                                                C  15100
C                                                                      C  15200
C   2. SET Y(1),Y(2),...,Y(NEQN) TO THE INITIAL VALUES TO BE           C  15300
C      USED FOR THE SOLUTION.                                          C  15400
C                                                                      C  15500
C   3. PROVIDE APPROPRIATE VALUES FOR NEQN, TINIT, TFINAL,             C  15600
C      TINCR, RELERR, ABSERR, AND IUNIT.  (SINCE THESE                 C  15700
C      QUANTITIES ARE NOT ALTERED BY STFINT, THEY MAY, IF              C  15800
C      DESIRED, BE WRITTEN AS CONSTANTS IN THE CALL LIST.)             C  15900
C                                                                      C  16000
C   4. PLACE THE ACTUAL NAMES OF FCT, FDER, AND OUTP IN THE            C  16100
C      CALL LIST, AND DECLARE THOSE NAMES IN AN EXTERNAL               C  16200
C      STATEMENT.                                                      C  16300
C                                                                      C  16400
C                                                                      C  16500
C                                                                      C  16600
C AT EACH CALL TO OUTP(T,Y,W), THE CURRENT VALUES OF THE               C  16700
C DERIVATIVES DY(I)/DT ARE IN W(1),W(2),...,W(NEQN).  THE              C  16800
C FIRST CALL TO OUTP IS MADE AT TINIT.                                 C  16900
C                                                                      C  17000
C IT IS NOT NECESSARY THAT THE INTEGRATION INTERVAL BE EVENLY          C  17100
C DIVISIBLE BY THE OUTPUT MESH SIZE TINCR.  HOWEVER, IF IT IS          C  17200
C NOT, INTEGRATION WILL PROCEED TO THE FIRST MESH POINT                C  17300
C BEYOND TFINAL, RATHER THAN STOPPING AT TFINAL.                       C  17400
C                                                                      C  17500
C**********************************************************************C  17600
C                                                                      C  17700
C THIS SUBROUTINE IS WRITTEN IN SINGLE PRECISION.                      C  17800
C A DOUBLE PRECISION VERSION FOR USE WITH A DOUBLE PRECISION           C  17900
C VERSION OF STIFF MAY BE OBTAINED SIMPLY BY REMOVING THE C            C  18000
C FROM COLUMN 1 OF THE NEXT FIVE CARDS.                                C  18100
C     DOUBLE PRECISION ABS,ABSER,ABSERR,DABS,DSIGN,DT,DTSGN,              18200
C    1                 D1,D2,EPSDT,RELER,RELERR,SIGN,T,TFINAL,            18300
C    2                 TINCR,TINIT,TMAXIN,TOUT,W,Y                        18400
C     ABS(D1) = DABS(D1)                                                  18500
C     SIGN(D1,D2) = DSIGN(D1,D2)                                          18600
C                                                                      C  18700
      COMMON /SFDATA/ T,TOUT,RELER,ABSER,DTSGN,NEQ,IFLAG,IOLD,            18800
```

```
     1                INIT,KOP
C                                                                C
      DIMENSION Y(NEQN),IW(NEQN),W(1)
C                                                                C
C  THE ACTUAL DIMENSION OF W, AS DEFINED BY THE CALLING PROGRAM, C
C  MUST BE AT LEAST NEQN**2+17*NEQN.                             C
C                                                                C
C                                                                C
      EXTERNAL FCT,FDER
C                                                                C
C  WRITE STARTING DATA INTO MESSAGE FILE                         C
C                                                                C
      IF (IUNIT.GT.0) WRITE(IUNIT,100) NEQN,TINIT,TFINAL,TINCR,
     1                                       RELERR,ABSERR
C                                                                C
C  COMPUTE INDICES FOR SPLITTING THE WORK ARRAY W(*).            C
C                                                                C
      K1=NEQN+1
      K2=K1+9*NEQN
      K3=K2+7*NEQN
C                                                                C
C  INITIALIZE VARIABLES TO START INTEGRATION WITH STIFF.         C
C                                                                C
      NEQ=NEQN
      RELER=RELERR
      ABSER=ABSERR
      IFLAG=1
      T=TINIT
      TOUT=TINIT
C                                                                C
C  SAVE INFORMATION NEEDED TO LOCATE END OF INTEGRATION.         C
C                                                                C
      TMAXIN=ABS(TFINAL-TINIT)
      EPSDT=0.001*ABS(TINCR)
C                                                                C
C  SET SIGN OF OUTPUT INTERVAL TO PROCEED FROM TINIT TO TFINAL.  C
C                                                                C
      DT=SIGN(TINCR,TFINAL-TINIT)
C                                                                C
C  INITIALIZE COUNTER OF ERROR RETURNS.                          C
C                                                                C
      IER=0
C                                                                C
C****************************************************************C
C                                                                C
C  CALL STIFF TO INTEGRATE TO THE NEXT OUTPUT POINT.             C
C                                                                C
    5 CALL STIFF(FCT,FDER,Y,W(1),IW,W(K2),W(K3),W(K1))
      IF (IFLAG.NE.2) GO TO 8
C                                                                C
C  SUCCESSFUL INTEGRATION TO OUTPUT POINT.                       C
C                                                                C
```

```
      CALL OUTP(T,Y,W)
C                                                                C
C  CHECK WHETHER TFINAL HAS BEEN REACHED OR PASSED.              C
C                                                                C
      IF (ABS(T-TINIT).GE.TMAXIN) GO TO 95
C                                                                C
C  INTEGRATION IS NOT YET COMPLETE, SO THE NEXT OUTPUT POINT IS  C
C  SET.  IF THE NEW POINT DIFFERS FROM TFINAL BY AN AMOUNT       C
C  ATTRIBUTABLE TO ROUNDOFF ERROR, IT IS RESET TO TFINAL EXACTLY.C
C  INTEGRATION IS THEN CONTINUED.                                C
C                                                                C
      TOUT=T+DT
      IF (ABS(TFINAL-TOUT).LT.EPSDT) TOUT=TFINAL
      GO TO 5
C                                                                C
C****************************************************************C
C                                                                C
C  INTEGRATION WAS INTERRUPTED BEFORE REACHING THE OUTPUT POINT. C
C  INCREMENT THE ERROR COUNTER AND BRANCH FOR WRITING THE        C
C  APPROPRIATE DIAGNOSTIC MESSAGE.  INTEGRATION WILL THEN BE     C
C  CONTINUED IF THE ERROR IS NOT TOO SEVERE.                     C
C                                                                C
    8 IER=IER+1
      GO TO (20,20,30,40,50,60,70,80,90),IFLAG
C                                                                C
C                                                                C
C  IFLAG HAS AN ILLEGAL VALUE, SO THAT A PRECISE DEFINITION OF   C
C  THE ERROR IS NOT POSSIBLE, AND EXECUTION MUST BE TERMINATED.  C
C                                                                C
   20 IF (IUNIT.GT.0) WRITE(IUNIT,200) T,IFLAG
      STOP
C                                                                C
C                                                                C
C  IFLAG = 3 -- THE REQUESTED ERROR TOLERANCES WERE TOO          C
C  SMALL FOR THE MACHINE PRECISION AND HAVE BEEN INCREASED TO    C
C  MINIMUM PERMISSIBLE VALUES.                                   C
C                                                                C
   30 IF (IUNIT.GT.0) WRITE(IUNIT,300) T,RELER,ABSER
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C  IFLAG = 4 -- THE REQUESTED ERROR TOLERANCES WERE TOO          C
C  STRINGENT FOR THIS PROBLEM AND HAVE BEEN INCREASED TO         C
C  VALUES WHICH SHOULD BE ACCEPTABLE.                            C
C                                                                C
   40 IF (IUNIT.GT.0) WRITE(IUNIT,400) T,RELER,ABSER
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 5 -- THE COUNTER OF CALLS TO FCT WAS RESET AFTER      C
C  EXCEEDING THE LIMIT IMPOSED IN STIFF.                         C
```

```
C                                                                C
   50 IF (IUNIT.GT.0) WRITE(IUNIT,500) T
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 6 -- THE CONVERGENCE TEST IN GEAR COULD NOT BE MET    C
C  USING THE SMALLEST ALLOWABLE STEPSIZE.  THE ERROR TOLERANCES  C
C  HAVE BEEN INCREASED TO VALUES WHICH WILL ALLOW THE            C
C  CONVERGENCE TEST TO BE PASSED.                                C
C                                                                C
   60 IF (IUNIT.GT.0) WRITE(IUNIT,600) T,RELER,ABSER
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 7 -- THE MONITOR OF OUTPUT FREQUENCY WAS RESET        C
C  AFTER INDICATING THAT THE OUTPUT INTERVAL IS TOO SMALL        C
C  TO ALLOW EFFICIENT OPERATION OF STIFF.                        C
C                                                                C
   70 IF (IUNIT.GT.0) WRITE(IUNIT,700) T
      IF (IER.GT.4) GO TO 92
      GO TO 5
C                                                                C
C                                                                C
C  IFLAG = 8 -- FATAL ERROR -- A COMPONENT OF THE SOLUTION       C
C  VANISHED, MAKING A PURE RELATIVE ERROR TEST IMPOSSIBLE.       C
C  THE USER MUST SUPPLY AN APPROPRIATE NONZERO VALUE OF          C
C  ABSERR FOR SUCCESSFUL INTEGRATION.                            C
C                                                                C
   80 IF (IUNIT.GT.0) WRITE(IUNIT,800) T
      STOP
C                                                                C
C                                                                C
C  IFLAG = 9 -- FATAL ERROR -- AN ILLEGAL PARAMETER VALUE        C
C  WAS PASSED TO STIFF.  THE USER MUST LOCATE AND CORRECT THE    C
C  ERROR.                                                        C
C                                                                C
   90 IF (IUNIT.GT.0) WRITE(IUNIT,900) T,NEQ,RELER,ABSER,TINCR
      STOP
C                                                                C
C                                                                C
C  THE PROGRAM IS STOPPED BECAUSE THERE HAVE BEEN FIVE ERROR     C
C  RETURNS FROM STIFF, INDICATING SERIOUS PROBLEMS WHICH         C
C  REQUIRE INTERVENTION BY THE USER.                             C
C                                                                C
   92 IF (IUNIT.GT.0) WRITE(IUNIT,920) T
      STOP
C                                                                C
C                                                                C
C  THE INTEGRATION HAS BEEN COMPLETED TO OR BEYOND TFINAL.       C
C                                                                C
```

```
   95 IF (IUNIT.GT.0) WRITE(IUNIT,950) T
      RETURN
C                                                                 C
C*****************************************************************C
C                                                                 C
C  THE REMAINDER OF THE PROGRAM CONTAINS THE FORMAT STATEMENTS    C
C  FOR THE DIAGNOSTIC MESSAGES.                                   C
C                                                                 C
  100 FORMAT(36H STFINT HAS BEEN CALLED TO INTEGRATE,I5,
     1  10H EQUATIONS/18H FROM INITIAL TIME,G12.5/
     2  14H TO FINAL TIME,G12.5/
     3  24H WITH OUTPUT AT STEPS OF,G12.5/
     4  32H AND INITIAL ERROR TOLERANCES OF/
     5  9H RELERR =,G12.5,9H ABSERR =,G12.5/)
C                                                                 C
  200 FORMAT(4H ***,12(4H****)/
     1  37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2  52H *** BECAUSE AN UNIDENTIFIABLE ERROR HAS CAUSED  ***/
     3  52H *** STIFF TO RETURN WITH THE ILLEGAL FLAG VALUE ***/
     4  12H *** IFLAG =,I11,26X,3H***/4H ***,12(4H****))
C                                                                 C
  300 FORMAT(17H *WARNING--AT T =,G12.5,22H, THE ERROR TOLERANCES/
     1  51H PASSED TO STIFF WERE FOUND TO BE TOO SMALL FOR THE/
     2  48H MACHINE PRECISION.  THEY HAVE BEEN INCREASED TO/
     3  9H RELERR =,G12.5,13H AND ABSERR =,G12.5/
     4  25H TO CONTINUE INTEGRATION./)
C                                                                 C
  400 FORMAT(17H *WARNING--AT T =,G12.5,21H, STIFF WAS UNABLE TO/
     1  52H MEET THE REQUESTED ERROR TOLERANCES AT THE SMALLEST/
     2  52H ALLOWABLE STEPSIZE.  INTEGRATION IS CONTINUING WITH/
     3  37H THE TOLERANCES INCREASED TO RELERR =,G12.5/
     4  13H AND ABSERR =,G12.5/)
C                                                                 C
  500 FORMAT(17H *WARNING--AT T =,G12.5,24H, THE NUMBER OF FUNCTION/
     1  51H CALLS EXCEEDED THE MAXIMUM NUMBER, MAXNFE, ALLOWED/
     2  52H BY STIFF.  THE FUNCTION EVALUATION COUNTER HAS BEEN/
     3  51H REDUCED BY MAXNFE TO ALLOW INTEGRATION TO PROCEED./)
C                                                                 C
  600 FORMAT(17H *WARNING--AT T =,G12.5,22H, THE CONVERGENCE TEST/
     1  41H IN GEAR COULD NOT BE MET AT THE SMALLEST/
     2  47H ALLOWABLE STEPSIZE.  INTEGRATION IS CONTINUING/
     3  48H WITH THE ERROR TOLERANCES INCREASED TO RELERR =/
     4  1H ,G12.5,13H AND ABSERR =,G12.5/)
C                                                                 C
  700 FORMAT(17H *WARNING--AT T =,G12.5,24H, STIFF RETURNED WITH AN/
     1  50H INDICATION THAT THIS INTEGRATION IS UNNECESSARILY/
     2  48H COSTLY BECAUSE THE REQUESTED OUTPUT INTERVAL IS/
     3  49H SUBSTANTIALLY SMALLER THAN THE NATURAL STEPSIZE./
     4  48H (REFER TO PROGRAM DOCUMENTATION FOR ALTERNATIVE/
     5  52H APPROACHES.)  THE OUTPUT FREQUENCY MONITOR HAS BEEN/
     6  39H RESET TO ALLOW INTEGRATION TO PROCEED./)
C                                                                 C
```

```
  800 FORMAT(4H ***,12(4H****)/
     1  37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2  40H *** BECAUSE A COMPONENT OF THE SOLUTION,9X,3H***/
     3  52H *** VANISHED WHILE ABSERR = 0.  IT WILL BE      ***/
     4  52H *** NECESSARY TO PROVIDE A NONZERO ABSOLUTE     ***/
     5  52H *** ERROR TOLERANCE TO PERFORM THIS INTEGRATION.***/
     6  4H ***,12(4H****))
C                                                                C
  900 FORMAT(4H ***,12(4H****)/
     1  37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2  52H *** BECAUSE INVALID INPUT WAS RECEIVED BY STIFF.***/
     3  37H *** CHECK THE FOLLOWING PARAMETERS--,12X,3H***/
     4  12H ***  NEQN =,I11,26X,3H***/
     5  14H ***  RELERR =,G12.5,23X,3H***/
     6  14H ***  ABSERR =,G12.5,23X,3H***/
     7  13H ***  TINCR =,G12.5,24X,3H***/
     8  52H *** ALSO, CHECK THAT THE OUTPUT ROUTINE DOES    ***/
     9  41H *** NOT RESET IFLAG TO AN ILLEGAL VALUE.,8X,3H***/
     A  4H ***,12(4H****))
C                                                                C
  920 FORMAT(4H ***,12(4H****)/
     1  37H *** THE PROGRAM IS TERMINATED AT T =,G12.5,3H***/
     2  52H *** BECAUSE STIFF HAS RETURNED IRREGULARLY FIVE ***/
     3  52H *** TIMES, WHICH INDICATES SOME SORT OF EXTREME ***/
     4  52H *** DIFFICULTY IN THE INTEGRATION.  THE USER    ***/
     5  52H *** SHOULD CAREFULLY CHECK THE ACCURACY OF HIS  ***/
     6  52H *** PROBLEM STATEMENT AND CODING.               ***/
     7  4H ***,12(4H****))
C                                                                C
  950 FORMAT(4H ***,12(4H****)/
     1  32H *** INTEGRATION COMPLETE AT T =,G12.5,5X,3H***/
     2  4H ***,12(4H****)///)
C                                                                C
C****************************************************************C
C                                                                C
C  END OF STFINT                                                 C
C                                                                C
      END


      SUBROUTINE STIFF(F,FDER,Y,YP,IPS,YY,W,WORK)
C                                                                C
C****************************************************************C
C                                                                C
C  SUBROUTINE STIFF USES SUBROUTINE GEAR TO INTEGRATE A SYSTEM   C
C  OF NEQN FIRST ORDER ORDINARY DIFFERENTIAL EQUATIONS           C
C                                                                C
C    DY/DT = F(T,Y) ,  Y = (Y(1),Y(2),...,Y(NEQN))               C
C                                                                C
C  FROM T TO TOUT.  FOR REASONS OF EFFICIENCY, STIFF INTEGRATES  C
C  BEYOND TOUT INTERNALLY, THOUGH NEVER BEYOND T + 10.*(TOUT-T), C
C  AND CALLS SUBROUTINE GINTRP TO INTERPOLATE THE SOLUTION AT    C
```

```
C  TOUT.  AN OPTION IS PROVIDED TO STOP THE INTEGRATION AT TOUT, C
C  BUT IT SHOULD BE USED ONLY IF THE CHARACTERISTICS OF THE      C
C  PROBLEM MAKE IT IMPOSSIBLE TO CONTINUE THE INTEGRATION BEYOND C
C  TOUT.                                                         C
C                                                                C
C  STIFF IS INTENDED FOR SOLVING STIFF PROBLEMS.  IT IS NOT      C
C  AN EFFICIENT MEANS FOR SOLVING NONSTIFF PROBLEMS.             C
C                                                                C
C  THE PARAMETERS IN THE ARGUMENT LIST ARE                       C
C                                                                C
C    F -- THE NAME OF THE USER-SUPPLIED SUBROUTINE F(T,Y,YP),    C
C         WHICH EVALUATES THE DERIVATIVES YP(I)=DY(I)/DT,        C
C         I=1,2,...,NEQN.                                        C
C    FDER -- THE NAME OF THE USER SUPPLIED SUBROUTINE WHICH      C
C            EVALUATES THE JACOBIAN MATRIX AS DESCRIBED IN THE   C
C            COMMENTS TO GEAR.                                   C
C    Y -- AN ARRAY OF DIMENSION NEQN, WHICH CONTAINS THE VALUE   C
C         OF THE SOLUTION.                                       C
C    YP -- AN ARRAY OF DIMENSION NEQN, WHICH CONTAINS THE VALUE  C
C          OF THE DERIVATIVE DY/DT.                              C
C    IPS,YY,W -- AN INTEGER ARRAY OF DIMENSION NEQN, AND         C
C                ARRAYS OF DIMENSIONS (NEQN,7) AND (NEQN,NEQN),  C
C                RESPECTIVELY.  THEIR VALUES ARE NEEDED FOR      C
C                CONTINUING INTEGRATION ON SUBSEQUENT CALLS.     C
C    WORK -- AN ARRAY OF DIMENSIONS (NEQN,9) WHICH IS USED       C
C            FOR WORKING STORAGE BY GEAR.                        C
C                                                                C
C  THE PARAMETERS IN THE COMMON BLOCK SFDATA ARE                 C
C                                                                C
C    T -- THE INDEPENDENT VARIABLE                               C
C                                                                C
C    TOUT -- THE VALUE OF T AT WHICH OUTPUT IS DESIRED           C
C                                                                C
C    RELERR,ABSERR -- RELATIVE AND ABSOLUTE ERROR TOLERANCES FOR C
C       THE LOCAL ERROR TEST.  EACH STEPSIZE IS CHOSEN SO THAT   C
C       ABS(LOCAL ERROR(I)) .LE. RELERR*ABS(Y(I))+ABSERR FOR     C
C       EACH COMPONENT OF THE SOLUTION.                          C
C                                                                C
C    NEQN -- THE NUMBER OF EQUATIONS                             C
C                                                                C
C    IFLAG -- INDICATOR OF THE STATUS OF INTEGRATION             C
C                                                                C
C    DTSGN,IOLD,INIT,KOP -- THOSE QUANTITIES USED FOR            C
C       INTERNAL BOOKKEEPING BY STIFF WHICH ARE NEEDED FOR       C
C       CONTINUING INTEGRATION, BUT WHICH ARE NORMALLY OF NO     C
C       INTEREST TO THE USER.  THEY ARE PLACED IN THE COMMON     C
C       BLOCK TO AVOID LOCAL RETENTION BY STIFF BETWEEN CALLS.   C
C       SEE THE PROGRAM DOCUMENTATION FOR A DESCRIPTION OF THEIR C
C       FUNCTIONING.                                             C
C                                                                C
C----------------------------------------------------------------C
C                                                                C
```

```
C  TO USE STIFF, THE CALLING PROGRAM MUST                         C
C                                                                 C
C    1. ALLOCATE SPACE FOR THE ARRAYS Y(NEQN), YP(NEQN),          C
C       IPS(NEQN), YY(NEQN,7), W(NEQN,NEQN), AND WORK(NEQN,9).    C
C       (Y AND YP MUST BE ARRAYS EVEN IF NEQN=1.)                 C
C                                                                 C
C    2. DECLARE THE NAMES OF F AND FDER IN AN EXTERNAL            C
C       STATEMENT, AND PLACE THOSE NAMES IN THE CALL LIST.        C
C                                                                 C
C                                                                 C
C  FIRST CALL TO STIFF                                            C
C  -------------------                                            C
C                                                                 C
C  TO BEGIN INTEGRATION, THE CALLING PROGRAM MUST                 C
C                                                                 C
C    1. INITIALIZE THE FOLLOWING VARIABLES IN THE COMMON BLOCK    C
C       SFDATA                                                    C
C         T -- THE STARTING VALUE OF THE INDEPENDENT VARIABLE     C
C         TOUT -- THE VALUE OF THE INDEPENDENT VARIABLE AT WHICH C
C            OUTPUT IS DESIRED.  (TOUT=T IS PERMITTED ON THE      C
C            FIRST CALL ONLY, IN WHICH CASE STIFF EVALUATES THE   C
C            DERIVATIVES YP AND RETURNS WITH IFLAG=2.)            C
C         RELERR -- THE DESIRED RELATIVE ERROR TOLERANCE          C
C         ABSERR -- THE DESIRED ABSOLUTE ERROR TOLERANCE          C
C         IFLAG -- NORMALLY +1.  IF THE INTEGRATION CANNOT        C
C                  PROCEED BEYOND TOUT, SET IFLAG = -1.           C
C                                                                 C
C    2. INITIALIZE Y(*) TO CONTAIN THE STARTING VALUE OF THE      C
C       SOLUTION.                                                 C
C                                                                 C
C                                                                 C
C  UPON SUCCESSFUL RETURN (IFLAG=2), T=TOUT, Y CONTAINS THE       C
C  SOLUTION AT TOUT, AND YP CONTAINS THE DERIVATIVES AT TOUT.     C
C                                                                 C
C  OTHERWISE, T IS THE LAST VALUE OF THE INDEPENDENT VARIABLE     C
C  SUCCESSFULLY REACHED DURING THE INTEGRATION, Y CONTAINS THE    C
C  SOLUTION AT T, YP CONTAINS THE DERIVATIVES AT T, AND IFLAG     C
C  INDICATES THE STATUS OF THE INTEGRATION AS DESCRIBED BELOW.    C
C                                                                 C
C                                                                 C
C  SUBSEQUENT CALLS TO STIFF                                      C
C  -------------------------                                      C
C                                                                 C
C  STIFF NORMALLY RETURNS WITH ALL INFORMATION NEEDED TO CONTINUEC
C  INTEGRATION.                                                   C
C                                                                 C
C  IF THE INTEGRATION REACHED TOUT (IFLAG=2), THE USER NEED       C
C  ONLY DEFINE A NEW TOUT AND CALL STIFF AGAIN.  IF THE INTE-     C
C  GRATION CANNOT CONTINUE INTERNALLY BEYOND THE NEW TOUT,        C
C  THE USER MUST ALSO SET IFLAG=-2.  IF THE NEW TOUT REPRESENTS   C
C  A REVERSAL OF THE DIRECTION OF INTEGRATION, IT IS NECESSARY    C
C  THAT Y NOT HAVE BEEN ALTERED.                                  C
```

```
C                                                                 C
C  IF THE INTEGRATION WAS INTERRUPTED WITH ABS(IFLAG) = 3, 4, 5, C
C  6, OR 7, INTEGRATION MAY BE CONTINUED SIMPLY BY CALLING STIFF C
C  AGAIN.  HOWEVER, SUCH A RETURN INDICATES THAT THE             C
C  INTEGRATION IS NOT PROCEEDING SO SMOOTHLY AS MIGHT BE         C
C  EXPECTED, SO THE USER SHOULD EXERCISE SOME CAUTION.           C
C                                                                 C
C  IF THE INTEGRATION WAS INTERRUPTED WITH IFLAG = 8, THE USER   C
C  MUST GIVE ABSERR A NONZERO POSITIVE VALUE AND SET IFLAG = 2   C
C  (OR IFLAG=-2 IF IT IS NECESSARY TO STOP AT TOUT) IN ORDER TO  C
C  CONTINUE.                                                     C
C                                                                 C
C  IF STIFF RETURNED WITH IFLAG = 9, THE USER MUST CORRECT THE   C
C  INVALID INPUT AND SET IFLAG = 2 OR -2 (OR IFLAG = 1 OR -1,    C
C  IF INTEGRATION HAS NOT YET STARTED) IN ORDER TO CONTINUE.     C
C                                                                 C
C----------------------------------------------------------------C
C                                                                 C
C  UPON RETURN, THE VALUE OF IFLAG INDICATES THE STATUS OF THE   C
C  INTEGRATION AS FOLLOWS,                                       C
C                                                                 C
C  IFLAG = 2 -- T=TOUT, SO THAT INTEGRATION WAS SUCCESSFUL TO .  C
C               THE OUTPUT POINT.                                C
C  IFLAG = 3 -- INTEGRATION WAS INTERRUPTED BECAUSE THE          C
C               REQUESTED ERROR TOLERANCES WERE TOO SMALL FOR    C
C               THE MACHINE PRECISION.  THEY HAVE BEEN INCREASED C
C               TO MINIMUM ACCEPTABLE VALUES.                    C
C  IFLAG = 4 -- INTEGRATION WAS INTERRUPTED BECAUSE THE          C
C               REQUESTED ERROR TOLERANCES COULD NOT BE          C
C               ATTAINED AT THE SMALLEST ALLOWABLE STEPSIZE.     C
C               THE TOLERANCES HAVE BEEN INCREASED TO VALUES     C
C               WHICH SHOULD ALLOW SUCCESSFUL CONTINUATION.      C
C  IFLAG = 5 -- INTEGRATION WAS INTERRUPTED AFTER THE COUNTER    C
C               OF CALLS TO F EXCEEDED MAXNFE.  THE COUNTER HAS  C
C               BEEN REDUCED BY MAXNFE.                          C
C  IFLAG = 6 -- INTEGRATION WAS INTERRUPTED BECAUSE THE          C
C               ERROR TOLERANCES WERE TOO SMALL TO PERMIT        C
C               THE ITERATION IN GEAR TO CONVERGE FOR THE        C
C               SMALLEST ALLOWABLE STEPSIZE.  THE TOLERANCES     C
C               HAVE BEEN INCREASED TO VALUES WHICH WILL         C
C               ALLOW CONVERGENCE.                               C
C  IFLAG = 7 -- INTEGRATION WAS NOT CONTINUED BECAUSE THE        C
C               SPACING OF OUTPUT POINTS HAS BEEN SO SMALL AS TO C
C               RESTRICT THE STEPSIZE FOR 50 CALLS TO GEAR,      C
C               INDICATING THAT THE CODE IS BEING USED IN A WAY  C
C               WHICH GREATLY IMPAIRS ITS EFFICIENCY.  THE       C
C               MONITOR OF THIS EFFECT WAS RESET TO ALLOW        C
C               CONTINUATION IF DESIRED.                         C
C  IFLAG = 8 -- INTEGRATION WAS INTERRUPTED WHEN A COMPONENT OF  C
C               THE SOLUTION VANISHED WHILE A PURE RELATIVE      C
C               ERROR TEST WAS SPECIFIED.  THE USER MUST SUPPLY  C
C               A NONZERO ABSERR TO CONTINUE INTEGRATION.        C
```

```
C  IFLAG = 9 -- INVALID INPUT WAS PASSED TO STIFF.  THE USER           C
C               MUST LOCATE AND CORRECT THE ERROR IN ORDER TO          C
C               CONTINUE INTEGRATION.                                  C
C  IFLAG =-3, -4, -5, -6, OR -7 -- SAME AS IFLAG = 3, 4, 5, 6,         C
C               OR 7, RESPECTIVELY, EXCEPT THAT STIFF WAS CALLED       C
C               WITH IFLAG NEGATIVE.                                   C
C                                                                      C
C**********************************************************************C
C                                                                      C
C  THIS SUBROUTINE IS WRITTEN IN SINGLE PRECISION.                     C
C  WITH MOST COMPILERS, A DOUBLE PRECISION VERSION, FOR USE WITH       C
C  DOUBLE PRECISION VERSIONS OF SUBROUTINES GEAR AND GINTRP, IS        C
C  OBTAINED SIMPLY BY SETTING THE APPROPRIATE VALUE FOR FOURU          C
C  IN THE DATA STATEMENT BELOW AND BY REMOVING THE C FROM              C
C  COLUMN 1 OF THE NEXT SEVEN CARDS.                                   C
C     DOUBLE PRECISION ABS,ABSDT,ABSEPS,ABSERR,AMAX1,DABS,DMAX1,
C    1                 DSIGN,DT,DTSGN,D1,D2,EPS,FOURU,H,HOLD,
C    2                 RELEPS,RELERR,RVAR,SIGN,T,TEND,TOUT,
C    3                 W,WORK,X,Y,YP,YY
C     ABS(D1) =DABS(D1)
C     SIGN(D1,D2) = DSIGN(D1,D2)
C     AMAX1(D1,D2) = DMAX1(D1,D2)
C                                                                      C
      LOGICAL START
C                                                                      C
      COMMON /SFDATA/ T,TOUT,RELERR,ABSERR,DTSGN,NEQN,IFLAG,
     1                IOLD,INIT,KOP
C                                                                      C
      COMMON /GEDATA/ X,HOLD,H,EPS,RVAR(8),NEQ,K,NFE,
     1                ICRASH,KOLD,ITST,IWEVAL,START
C                                                                      C
      DIMENSION Y(NEQN),YP(NEQN),IPS(NEQN),YY(NEQN,7),
     1          W(NEQN,NEQN),WORK(NEQN,9)
C                                                                      C
      EXTERNAL F,FDER
C                                                                      C
C                                                                      C
C  IN THE FOLLOWING DATA STATEMENT, FOURU MUST BE SET TO A             C
C  VALUE APPROPRIATE TO THE MACHINE BEING USED.                        C
C                                                                      C
C  FOURU IS 4.*U, WHERE THE COMPUTER UNIT ROUNDOFF ERROR U IS          C
C  THE SMALLEST POSITIVE VALUE REPRESENTABLE IN THE MACHINE            C
C  SUCH THAT (1.+U).GT.1.                                              C
C                                                                      C
C  SOME REPRESENTATIVE VALUES ARE                                      C
C    U = 9.54E-7        FOURU = 3.82E-6       IBM 360/370 S.P.         C
C    U = 5.96E-8        FOURU = 2.39E-7       PDP-11 S.P.              C
C    U = 1.49E-8        FOURU = 5.97E-8       UNIVAC 1108              C
C    U = 7.45E-9        FOURU = 2.99E-8       PDP-10                   C
C    U = 7.11E-15       FOURU = 2.85E-14      CDC 6000 SERIES          C
C    U = 2.22D-16       FOURU = 8.89D-16      IBM 360/370 D.P.         C
C    U = 1.39D-17       FOURU = 5.56D-17      PDP-11 D.P.              C
```

```
C                                                                C
      DATA FOURU/2.39E-7/
C                                                                C
C                                                                C
C  THE EXPENSE IS CONTROLLED BY RESTRICTING THE NUMBER          C
C  OF FUNCTION EVALUATIONS TO BE APPROXIMATELY MAXNFE.          C
C  AS SET, THIS CORRESPONDS TO ABOUT 1500 STEPS.                C
C                                                                C
      DATA MAXNFE/4000/
C                                                                C
C****************************************************************C
C                                                                C
C  TEST FOR IMPROPER PARAMETERS                                  C
C                                                                C
      MFLAG=IABS(IFLAG)
      IF ((MFLAG.GT.7).OR.(MFLAG.EQ.0).OR.(NEQN.LT.1)) GO TO 200
      IF ((RELERR.LT.0.0).OR.(ABSERR.LT.0.0)) GO TO 200
      EPS=AMAX1(RELERR,ABSERR)
      IF (EPS.LE.0.0) GO TO 200
C                                                                C
      NEQ=NEQN
      DT=TOUT-T
      ABSDT=ABS(DT)
      TEND=T + 10.*DT
      IF(IFLAG.LT.0) TEND=TOUT
      RELEPS=RELERR/EPS
      ABSEPS=ABSERR/EPS
C                                                                C
C  IF THIS IS THE FIRST CALL, GO TO INITIALIZATION SECTION.     C
C                                                                C
      IF (MFLAG.EQ.1) GO TO 150
C                                                                C
C  SINCE THIS IS NOT THE FIRST CALL, T=TOUT IS NOT VALID        C
C  INPUT.                                                        C
C                                                                C
      IF (T.EQ.TOUT) GO TO 200
C                                                                C
C  IF INITIALIZATION WAS NOT COMPLETED ON THE FIRST CALL, DO    C
C  IT NOW.                                                       C
C                                                                C
      IF (INIT.EQ.0) GO TO 180
C                                                                C
C  IF THE LAST STEP WAS MADE WITH IFLAG NEGATIVE, OR IF THE     C
C  DIRECTION OF INTEGRATION IS CHANGED, RESTART THE INTEGRATION. C
C                                                                C
      IF (IOLD .LT. 0) GO TO 160
      IF (DTSGN*DT .LT. 0.0) GO TO 160
C                                                                C
C  IF ALREADY PAST OUTPUT POINT, INTERPOLATE AND RETURN.        C
C                                                                C
   50 IF(ABS(X-T) .LT. ABSDT)GO TO 60
      CALL GINTRP(TOUT,Y,YP,YY)
      IOLD=IFLAG
      IFLAG=2
      T=TOUT
      RETURN
C                                                                 C
C  IF CANNOT GO PAST OUTPUT POINT, TEST WHETHER TOO CLOSE TO      C
C  INTEGRATE WITH GEAR.                                           C
C                                                                 C
   60 IF ((IFLAG.LT.0).AND.(ABS(TOUT-X).LT.FOURU*ABS(X))) GO TO 210
C                                                                 C
C  TEST FOR TOO MUCH WORK.                                        C
C                                                                 C
      IF(NFE .GT. MAXNFE) GO TO 230
C                                                                 C
C  LIMIT STEPSIZE, SET WEIGHT VECTOR, AND TAKE A STEP.            C
C                                                                 C
      IF (ABS(H).LT.ABS(TEND-X)) GO TO 100
      KOP=KOP+1
      IF (KOP.GT.50) GO TO 290
      H=TEND-X
C                                                                 C
C  THE WEIGHT VECTOR IS PLACED IN Y(*).                           C
C                                                                 C
  100 DO 110 L=1,NEQN
        Y(L)=RELEPS*ABS(YY(L,1))+ABSEPS
        IF (Y(L).EQ.0.0) GO TO 270
  110   CONTINUE
C                                                                 C
      CALL GEAR(F,FDER,YY,W,IPS,Y,YP,WORK(1,1),WORK(1,2),
     1          WORK(1,3))
C                                                                 C
C  TEST WHETHER STEP WAS SUCCESSFUL.                              C
C                                                                 C
      IF (ICRASH.NE.0) GO TO 250
C                                                                 C
C  STEP SUCCEEDED.  CONTINUE WITH INTEGRATION.                    C
C                                                                 C
      GO TO 50
C                                                                 C
C*****************************************************************C
C                                                                 C
C  INITIALIZATION SECTION --                                      C
C                                                                 C
C                                                                 C
C  ON START AND RESTART, SET WORK VARIABLES X AND YY, STORE       C
C  THE DIRECTION OF INTEGRATION, AND INITIALIZE THE STEPSIZE.     C
C                                                                 C
  150 NFE=0
  160 START=.TRUE.
      INIT=0
      KOP=0
      X=T
      DO 170 L=1,NEQN
  170   YY(L,1)=Y(L)
C                                                                C
C  IF T=TOUT ON FIRST CALL, EVALUATE DERIVATIVES AND RETURN.     C
C                                                                C
      IF (T.NE.TOUT) GO TO 180
      CALL F(X,YY,YP)
      NFE=NFE+1
      IFLAG=2
      RETURN
C                                                                C
C  SET STEPSIZE AND TRANSFER TO BEGIN INTEGRATION.               C
C                                                                C
  180 INIT=1
      DTSGN=DT
      H=SIGN(AMAX1(ABS(TOUT-X),FOURU*ABS(X)),TOUT-X)
      GO TO 50
C                                                                C
C****************************************************************C
C                                                                C
C  THE REMAINDER OF THE PROGRAM CONTAINS STATEMENTS TO HANDLE    C
C  VARIOUS EXCEPTIONAL OCCURRENCES.                              C
C                                                                C
C                                                                C
C  INVALID INPUT WAS PASSED TO STIFF.                            C
C                                                                C
  200 IFLAG=9
      RETURN
C                                                                C
C  CANNOT PASS TOUT, BUT TOO CLOSE TO CALL GEAR.  EXTRAPOLATE    C
C  BY EULER METHOD AND RETURN.                                   C
C                                                                C
  210 H=TOUT-X
      CALL F(X,YY,YP)
      NFE=NFE+1
      DO 220 L=1,NEQN
  220   Y(L)=YY(L,1)+H*YP(L)
      IFLAG=2
      T=TOUT
      CALL F(T,Y,YP)
      NFE=NFE+1
      IOLD=-1
      RETURN
C                                                                C
C  NUMBER OF FUNCTION CALLS EXCEEDED MAXNFE.  RESET COUNTER NFE. C
C                                                                C
  230 IFLAG=ISIGN(5,IFLAG)
      NFE=NFE-MAXNFE
      GO TO 300
C                                                                C
C  ERROR TOLERANCES WERE TOO SMALL TO PERMIT A SUCCESSFUL STEP.  C
C  INCREASE RELERR AND ABSERR, AND SET IFLAG TO INDICATE THE     C
```

```
C  CAUSE OF THE DIFFICULTY.                                     C
C                                                               C
  250 IFLAG=ISIGN(4,IFLAG)
      IF (ICRASH.EQ.2) IFLAG=ISIGN(3,IFLAG)
      IF (ICRASH.EQ.3) IFLAG=ISIGN(6,IFLAG)
      RELERR=EPS*RELEPS
      ABSERR=EPS*ABSEPS
      GO TO 300
C                                                               C
C  A COMPONENT OF THE SOLUTION VANISHED WHEN A PURE RELATIVE    C
C  ERROR TEST WAS SPECIFIED.  USER MUST SET ABSERR POSITIVE     C
C  IN ORDER TO PROCEED.                                         C
C                                                               C
  270 IFLAG=8
      GO TO 300
C                                                               C
C  THE OUTPUT INTERVAL HAS BEEN TOO SMALL TO ALLOW EFFICIENT    C
C  OPERATION OF STIFF.                                          C
C                                                               C
  290 KOP=0
      IFLAG=ISIGN(7,IFLAG)
C                                                               C
C  FOR ERROR RETURN, SET T, Y(*), AND YP(*) TO THE MOST RECENT  C
C  VALUES REACHED IN THE INTEGRATION.                           C
C                                                               C
  300 T=X
      IOLD=1
      DO 310 L=1,NEQN
        Y(L)=YY(L,1)
  310   YP(L)=YY(L,2)/HOLD
      RETURN
C                                                               C
C***************************************************************C
C                                                               C
C  END OF STIFF                                                 C
C                                                               C
      END


      SUBROUTINE GEAR(FCT,FDER,Y,W,IPS,WT,YP,C,ERROR,SAVE)
C                                                               C
C***************************************************************C
C                                                               C
C  SUBROUTINE GEAR USES A BACKWARD DIFFERENTIATION METHOD (OF   C
C  ORDER .LE. 6) TO PERFORM ONE STEP IN THE INTEGRATION OF A    C
C  SYSTEM OF NEQN FIRST ORDER ORDINARY DIFFERENTIAL EQUATIONS   C
C                                                               C
C    DY/DT = F(T,Y) ,  Y = (Y(1),Y(2),...,Y(NEQN))              C
C                                                               C
C  TO SPECIFY THE DIFFERENTIAL EQUATIONS TO BE SOLVED, THE USER C
C  MUST SUPPLY TWO SUBROUTINES.  THE FIRST, DENOTED HERE BY     C
C  FCT, CALCULATES THE NEQN DERIVATIVES DY(I)/DT.  THE SECOND,  C
```

```
C  DENOTED HERE BY FDER, CALCULATES THE NEQN**2 ELEMENTS OF       C       11300
C  THE JACOBIAN MATRIX OF F(T,Y).                                  C       11400
C                                                                  C       11500
C    FCT MUST BE OF THE FORM FCT(T,Y,YP), WHERE Y AND YP ARE       C       11600
C    ARRAYS EACH OF DIMENSION NEQN.  FCT MUST CALCULATE THE        C       11700
C    VALUES OF THE DERIVATIVES DY(I)/DT AT (T,Y) AND PLACE         C       11800
C    THEM IN YP.  T AND Y MUST NOT BE ALTERED BY FCT.              C       11900
C                                                                  C       12000
C    FDER MUST BE OF THE FORM FDER(T,Y,W,YP,DER), WHERE Y AND      C       12100
C    YP ARE ARRAYS EACH OF DIMENSION NEQN, AND W IS AN ARRAY       C       12200
C    OF DIMENSIONS (NEQN,NEQN).  DER IS A SUBROUTINE NAME.         C       12300
C    WHEN FDER IS CALLED, DER IS THE NAME WHICH THE USER HAS       C       12400
C    SUPPLIED FOR FCT.  FDER MUST CALCULATE THE VALUES OF ALL      C       12500
C    THE PARTIAL DERIVATIVES OF F AT (T,Y).  THE PARTIAL           C       12600
C    DERIVATIVE OF THE M-TH COMPONENT OF F WITH RESPECT TO Y(N)    C       12700
C    IS TO BE PLACED IN W(M,N).  WHEN FDER IS CALLED, YP           C       12800
C    CONTAINS THE VALUES OF DY/DT AT (T,Y).  FOR SOME PROBLEMS,    C       12900
C    THESE VALUES MAY BE USEFUL IN SIMPLIFYING THE EVALUATION      C       13000
C    OF THE JACOBIAN.  T, Y, AND YP MUST NOT BE ALTERED BY FDER.   C       13100
C                                                                  C       13200
C    FOR FDER, THE USER MAY SUPPLY THE DEFAULT SUBROUTINE          C       13300
C    QJACOB, WHICH USES FCT TO COMPUTE THE JACOBIAN BY             C       13400
C    NUMERICAL DIFFERENCING.  SINCE THIS PROCEDURE INVOLVES        C       13500
C    NEQN CALLS TO FCT, ITS USE IS NOT RECOMMENDED UNLESS THE      C       13600
C    JACOBIAN IS DIFFICULT TO COMPUTE DIRECTLY.                    C       13700
C                                                                  C       13800
C  SUBROUTINE GEAR IS INTENDED SPECIFICALLY FOR THE SOLUTION OF    C       13900
C  STIFF PROBLEMS.  FOR NONSTIFF PROBLEMS, GEAR WILL BE LESS       C       14000
C  EFFICIENT THAN A CODE BASED ON ADAMS METHODS.                   C       14100
C                                                                  C       14200
C  INFORMATION IS EXCHANGED WITH GEAR BOTH THROUGH THE ARGUMENT    C       14300
C  LIST AND THROUGH THE COMMON BLOCK GEDATA.                       C       14400
C                                                                  C       14500
C  THE PARAMETERS IN THE ARGUMENT LIST ARE                         C       14600
C                                                                  C       14700
C    FCT -- THE NAME OF THE USER-SUPPLIED SUBROUTINE TO            C       14800
C       CALCULATE THE DERIVATIVES                                  C       14900
C    FDER -- THE NAME OF THE USER-SUPPLIED SUBROUTINE TO           C       15000
C       CALCULATE THE JACOBIAN MATRIX                              C       15100
C    Y -- AN ARRAY OF DIMENSIONS (NEQN,7) WHOSE VALUE MUST NOT     C       15200
C       BE ALTERED BETWEEN CALLS TO GEAR.  THE FIRST NEQN          C       15300
C       LOCATIONS, Y(1,1),Y(2,1),...,Y(NEQN,1) CONTAIN THE         C       15400
C       SOLUTION VALUES                                            C       15500
C    W,IPS -- RESPECTIVELY, AN ARRAY OF DIMENSIONS (NEQN,NEQN)     C       15600
C       AND AN INTEGER ARRAY OF DIMENSION NEQN, WHOSE VALUES       C       15700
C       MUST NOT BE ALTERED BETWEEN CALLS TO GEAR                  C       15800
C    WT -- AN ARRAY OF DIMENSION NEQN USED IN SPECIFYING THE       C       15900
C       ERROR TEST.                                                C       16000
C    YP,C,ERROR,SAVE -- RESPECTIVELY, ARRAYS OF DIMENSIONS NEQN,   C       16100
C       NEQN, NEQN, AND (NEQN,7), WHICH ARE USED FOR WORKING       C       16200
C       STORAGE BY GEAR.  IT IS NOT NECESSARY TO PRESERVE          C       16300
C       THEIR VALUES BETWEEN STEPS                                 C       16400
```

```
C                                                                 C
C  THE PARAMETERS IN THE COMMON BLOCK GEDATA ARE                  C
C                                                                 C
C    T -- THE INDEPENDENT VARIABLE                                C
C    HOLD -- THE STEPSIZE USED FOR THE LAST SUCCESSFUL STEP       C
C    H -- THE STEPSIZE TO BE USED IN ATTEMPTING THE NEXT STEP     C
C    EPS -- THE DESIRED TOLERANCE FOR THE ERROR TEST              C
C    NEQN -- THE NUMBER OF EQUATIONS BEING INTEGRATED             C
C    K -- THE ORDER OF THE METHOD TO BE USED FOR THE NEXT STEP    C
C    ICRASH -- AN INDICATOR OF ERROR CONDITIONS UPON RETURN       C
C    NFE -- A COUNTER WHICH IS INCREMENTED BY 1 AFTER EACH        C
C       CALL TO FCT                                               C
C    START -- A LOGICAL VARIABLE WHICH MUST BE SET .TRUE. ON      C
C       THE FIRST CALL TO GEAR                                    C
C    AHINV,A(7),KOLD,ITST,IWEVAL -- VARIABLES USED FOR INTERNAL   C
C       BOOKKEEPING BY GEAR WHOSE VALUES ARE NEEDED FOR           C
C       CONTINUING INTEGRATION.  THEIR VALUES ARE NORMALLY OF     C
C       NO INTEREST TO THE USER, AND THEY ARE PLACED IN THE       C
C       COMMON BLOCK ONLY TO AVOID LOCAL RETENTION BETWEEN        C
C       CALLS.                                                    C
C                                                                 C
C  EXCEPT FOR H, EPS, AND NFE, THE VALUES IN THE COMMON BLOCK     C
C  MUST NOT BE ALTERED BETWEEN CALLS TO GEAR.                     C
C                                                                 C
C  THE COUNTER NFE IS PROVIDED SO THAT THE USER CAN MONITOR THE   C
C  AMOUNT OF WORK BEING DONE.  ITS VALUE HAS NO MEANING TO GEAR.  C
C                                                                 C
C-----------------------------------------------------------------C
C                                                                 C
C  GEAR AUTOMATICALLY ADJUSTS THE STEPSIZE AND THE ORDER OF THE   C
C  METHOD SO THAT THE MAX NORM OF THE VECTOR WITH COMPONENTS      C
C  LOCAL ERROR(I)/WT(I) IS LESS THAN EPS FOR EACH STEP.  THE      C
C  ARRAY WT ALLOWS THE USER TO SPECIFY AN ERROR TEST WHICH IS     C
C  APPROPRIATE FOR HIS PROBLEM.                                   C
C                                                                 C
C  FOR EXAMPLE,                                                   C
C                                                                 C
C    WT(I) = 1.0  SPECIFIES ABSOLUTE ERROR                        C
C                                                                 C
C    WT(I) = ABS(Y(I,1)) SPECIFIES ERROR RELATIVE TO THE MOST     C
C                 RECENT VALUE OF THE I-TH COMPONENT OF THE       C
C                 SOLUTION                                        C
C                                                                 C
C    WT(I) = AMAX1(WT(I),ABS(Y(I,1))) SPECIFIES ERROR RELATIVE    C
C                 TO THE LARGEST MAGNITUDE YET ACHIEVED BY THE    C
C                 I-TH COMPONENT OF THE SOLUTION                  C
C                                                                 C
C    WT(I) = ABS(Y(I,1))*RELERR/EPS+ABSERR/EPS, WHERE EPS =       C
C                 AMAX1(RELERR,ABSERR), SPECIFIES A MIXED ERROR   C
C                 TEST WHERE RELERR IS RELATIVE ERROR AND         C
C                 ABSERR IS ABSOLUTE ERROR                        C
C                                                                 C
```

```
C  ALL COMPONENTS OF WT MUST BE NONZERO.                         C
C                                                                C
C----------------------------------------------------------------C
C                                                                C
C  UPON RETURN, THE VALUE OF ICRASH INDICATES THE ACTION WHICH   C
C  WAS TAKEN.                                                    C
C                                                                C
C    ICRASH = 0 -- A STEP OF LENGTH HOLD WAS SUCCESSFULLY        C
C       TAKEN TO THE VALUE NOW IN T.  THE SOLUTION AT T          C
C       IS IN Y(*,1), AND Y(*,2) CONTAINS THE DERIVATIVES        C
C       MULTIPLIED BY HOLD.                                      C
C                                                                C
C    ICRASH = 1 -- NO STEP WAS TAKEN BECAUSE H WAS SMALLER THAN  C
C       PERMITTED BY THE MACHINE PRECISION.  H WAS INCREASED     C
C       TO AN ACCEPTABLE VALUE.                                  C
C                                                                C
C    ICRASH = 2 -- NO STEP WAS TAKEN BECAUSE EPS WAS TOO SMALL   C
C       FOR THE MACHINE PRECISION.  EPS WAS INCREASED TO AN      C
C       ACCEPTABLE VALUE.                                        C
C                                                                C
C    ICRASH = 3 -- NO STEP WAS TAKEN BECAUSE CORRECTOR           C
C       CONVERGENCE COULD NOT BE ACHIEVED AT THE SMALLEST        C
C       ALLOWED STEPSIZE.  EPS IS INCREASED TO A VALUE WHICH     C
C       ALLOWS THE CONVERGENCE TEST TO BE PASSED.                C
C                                                                C
C    ICRASH = 4 -- NO STEP WAS TAKEN BECAUSE THE ERROR TEST      C
C       COULD NOT BE SATISFIED AT THE SMALLEST ALLOWED STEPSIZE. C
C       EPS IS INCREASED TO A VALUE WHICH IS ACHIEVABLE.         C
C                                                                C
C----------------------------------------------------------------C
C                                                                C
C  FIRST CALL TO GEAR                                            C
C  ------------------                                            C
C                                                                C
C  TO BEGIN THE SOLUTION, THE CALLING PROGRAM MUST               C
C                                                                C
C    1. PROVIDE STORAGE FOR THE ARRAYS Y(NEQN,7), W(NEQN,NEQN),  C
C       IPS(NEQN), WT(NEQN), YP(NEQN), C(NEQN), ERROR(NEQN),     C
C       AND SAVE(NEQN,7)                                         C
C                                                                C
C    2. PLACE THE INITIAL VALUES OF THE SOLUTION IN              C
C       Y(1,1), Y(2,1), ..., Y(NEQN,1).                          C
C                                                                C
C    3. SET WT(1), ..., WT(NEQN) TO NONZERO VALUES TO PROVIDE    C
C       THE ERROR TEST DESIRED.                                  C
C                                                                C
C    4. INITIALIZE THE FOLLOWING VARIABLES IN THE COMMON BLOCK  C
C       GEDATA                                                   C
C          T -- THE STARTING VALUE OF THE INDEPENDENT VARIABLE  C
C          H -- A NOMINAL STEPSIZE FOR THE FIRST STEP,           C
C             INDICATING THE DIRECTION OF INTEGRATION AND THE    C
C             MAXIMUM STEPSIZE.  H WILL BE REDUCED AS NECESSARY C
```

```
C             TO MEET THE ERROR CRITERION.                      C
C          EPS -- THE DESIRED ERROR TOLERANCE                   C
C          NEQN -- THE NUMBER OF EQUATIONS                      C
C          NFE -- SET TO 0 TO BEGIN A COUNT OF FUNCTION CALLS   C
C          START -- SET .TRUE.                                  C
C                                                               C
C    4. PLACE THE ACTUAL NAMES OF THE USER SUPPLIED SUBROUTINES C
C       FCT AND FDER IN THE CALL LIST, AND DECLARE THOSE NAMES  C
C       IN AN EXTERNAL STATEMENT.                               C
C                                                               C
C                                                               C
C  SUBSEQUENT CALLS TO GEAR                                     C
C  ------------------------                                     C
C                                                               C
C  GEAR RETURNS WITH ALL INFORMATION NEEDED TO CONTINUE THE     C
C  INTEGRATION.  TO MAINTAIN RELATIVE ERROR TESTS AS DESCRIBED  C
C  ABOVE, WT(*) MUST BE UPDATED AFTER EACH STEP.  CALLING GEAR  C
C  AGAIN THEN ADVANCES THE SOLUTION ANOTHER STEP.  TO CONTINUE  C
C  INTEGRATION USING THE REVISED STEPSIZE OR TOLERANCE          C
C  PROVIDED WITH AN ERROR RETURN (ICRASH.NE.0), IT IS ONLY      C
C  NECESSARY TO CALL GEAR AGAIN.                                C
C                                                               C
C  NORMALLY, THE INTEGRATION IS CONTINUED TO THE FIRST STEP     C
C  BEYOND THE DESIRED OUTPUT POINT, AND THE SOLUTION IS INTER-  C
C  POLATED TO THE OUTPUT POINT USING THE SUBROUTINE GINTRP.     C
C  IF IT IS IMPOSSIBLE TO INTEGRATE BEYOND THE OUTPUT POINT, H  C
C  MAY BE REDUCED TO HIT THE OUTPUT POINT.  OTHERWISE, H SHOULD C
C  NOT BE ALTERED, SINCE CHANGING H IMPAIRS THE EFFICIENCY OF   C
C  THE CODE.  H MUST NEVER BE INCREASED.  EPS MAY BE CHANGED    C
C  BETWEEN CALLS, BUT LARGE DECREASES SHOULD BE AVOIDED, SINCE  C
C  THIS WILL INVALIDATE THE STEP SELECTION MECHANISM AND LEAD   C
C  TO STEP FAILURES.                                            C
C                                                               C
C***************************************************************C
C                                                               C
C  THIS SUBROUTINE IS WRITTEN IN SINGLE PRECISION.              C
C  WITH MOST COMPILERS, A DOUBLE PRECISION VERSION MAY BE       C
C  OBTAINED SIMPLY BY PROVIDING A DOUBLE PRECISION VERSION OF   C
C  DECOMP, BY SETTING THE APPROPRIATE VALUE FOR FOURU IN THE    C
C  DATA STATEMENT BELOW, AND BY REMOVING THE C FROM COLUMN 1 OF C
C  THE NEXT TEN CARDS.                                          C
C     DOUBLE PRECISION A,ABS,ABSH,ADEN,AHINV,AMAX1,ANUM,BND,C,
C    1                 CNORM,D,DABS,DMAX1,DSIGN,DSQRT,DUM1,DUM2,
C    2                 D1,EPS,EPSMIN,ERDWN,ERROR,ERTST,ERUP,
C    3                 FOURU,GFACT,H,HMIN,HOLD,R,RATIO,RDWN,RUP,
C    4                 R1,SAVE,SIGN,SQRT,SUM,T,TOLD,W,WT,Y,
C    5                 YNORM,YP,YPNORM
C     ABS(DUM1)=DABS(DUM1)
C     AMAX1(DUM1,DUM2)=DMAX1(DUM1,DUM2)
C     SIGN(DUM1,DUM2)=DSIGN(DUM1,DUM2)
C     SQRT(DUM1)=DSQRT(DUM1)
C                                                               C
```

```
      LOGICAL START
C                                                                 C
      EXTERNAL FCT
C                                                                 C
      COMMON /GEDATA/ T,HOLD,H,EPS,AHINV,A(7),NEQN,K,NFE,
     1                ICRASH,KOLD,ITST,IWEVAL,START
C                                                                 C
      DIMENSION Y(NEQN,7),SAVE(NEQN,7),YP(NEQN),WT(NEQN),
     1          ERROR(NEQN),C(NEQN),IPS(NEQN),W(NEQN,NEQN),
     2          ERTST(6),ERUP(5),ERDWN(6),ANUM(27),ADEN(6)
C                                                                 C
      DATA ERTST/2.0,4.5,7.333,10.42,13.7,17.15/
      DATA ERUP/3.0,6.0,9.167,12.5,15.98/
      DATA ERDWN/1.0,1.0,0.5,0.1667,0.04133,0.008267/
C                                                                 C
      DATA ANUM/1.,1.,2.,3.,1.,6.,11.,6.,1.,24.,50.,35.,10.,
     1          1.,120.,274.,225.,85.,15.,1.,720.,1764.,
     2          1624.,735.,175.,21.,1./
      DATA ADEN/-1.,-3.,-11.,-50.,-274.,-1764./
C                                                                 C
C                                                                 C
C  IN THE FOLLOWING DATA STATEMENT, FOURU MUST BE SET TO A        C
C  VALUE APPROPRIATE TO THE MACHINE BEING USED.                   C
C                                                                 C
C  FOURU IS 4.*U, WHERE THE COMPUTER UNIT ROUNDOFF ERROR U IS     C
C  THE SMALLEST POSITIVE VALUE REPRESENTABLE IN THE MACHINE       C
C  SUCH THAT (1.+U).GT.1.                                         C
C                                                                 C
C  SOME REPRESENTATIVE VALUES ARE                                 C
C    U = 9.54E-7      FOURU = 3.82E-6      IBM 360/370 S.P.       C
C    U = 5.96E-8      FOURU = 2.39E-7      PDP-11 S.P.            C
C    U = 1.49E-8      FOURU = 5.97E-8      UNIVAC 1108            C
C    U = 7.45E-9      FOURU = 2.99E-8      PDP-10                 C
C    U = 7.11E-15     FOURU = 2.85E-14     CDC 6000 SERIES        C
C    U = 2.22D-16     FOURU = 8.89D-16     IBM 360/370 D.P.       C
C    U = 1.39D-17     FOURU = 5.56D-17     PDP-11 D.P.            C
C                                                                 C
      DATA FOURU/2.39E-7/
C                                                                 C
C*****************************************************************C
C                 ******                                          C
C  BEGIN BLOCK 0  ******                                          C
C                 ******                                          C
C**********************                                           C
C                                                                 C
C  IF STEPSIZE IS TOO SMALL FOR MACHINE PRECISION, INCREASE IT    C
C  TO AN ACCEPTABLE VALUE AND RETURN TO CALLING PROGRAM.          C
C                                                                 C
      HMIN=FOURU*ABS(T)
      IF (ABS(H).GE.HMIN) GO TO 5
      H=SIGN(HMIN,H)
      ICRASH=1
      RETURN
C                                                                C
C  IF THE REQUESTED ERROR TOLERANCE IS TOO SMALL FOR THE MACHINE C
C  PRECISION, INCREASE IT TO AN ACCEPTABLE VALUE AND RETURN TO   C
C  THE CALLING PROGRAM.                                          C
C                                                                C
    5 YNORM=0.0
      DO 10 I=1,NEQN
   10   YNORM=AMAX1(YNORM,ABS(Y(I,1)/WT(I)))
      EPSMIN=FOURU*YNORM
      IF (EPS.GE.EPSMIN) GO TO 15
      EPS=EPSMIN*(1.+FOURU)
      ICRASH=2
      RETURN
C                                                                C
C  IF THIS IS THE FIRST STEP, CHOOSE A STARTING STEPSIZE AND     C
C  INITIALIZE Y(*,2).                                            C
C                                                                C
   15 IF (.NOT.START) GO TO 50
      CALL FCT(T,Y,YP)
      NFE=NFE+1
      YPNORM=0.0
      DO 20 I=1,NEQN
   20   YPNORM=AMAX1(YPNORM,ABS(YP(I)/WT(I)))
      ABSH=ABS(H)
      IF (EPS.LT.16.*YPNORM*H**2) ABSH=0.25*SQRT(EPS/YPNORM)
      H=SIGN(AMAX1(ABSH,HMIN),H)
      DO 25 I=1,NEQN
   25   Y(I,2)=YP(I)*H
      HOLD=H
      K=1
      KOLD=0
      START=.FALSE.
C                                                                C
C  INITIALIZE CONTROL VARIABLES FOR THIS STEP.  SAVE T AND Y     C
C  FOR USE IN CASE OF STEP FAILURE.                              C
C                                                                C
   50 ICRASH=0
      IFAIL=0
      KP1=K+1
      TOLD=T
      DO 60 J=1,KP1
        DO 60 I=1,NEQN
   60     SAVE(I,J)=Y(I,J)
C                                                                C
C***********************                                         C
C                 ******                                         C
C  END BLOCK 0    ******                                         C
C                 ******                                         C
C****************************************************************C
C                 ******                                         C
C  BEGIN BLOCK 1  ******                                         C
```

```
C                 ******                                          C
C***********************                                          C
C                                                                 C
C  IF ORDER HAS BEEN CHANGED, RECOMPUTE COEFFICIENTS A(*)         C
C                                                                 C
  100 IF (K.EQ.KOLD) GO TO 120
      KP1=K+1
      INDEX=(K*KP1)/2 - 1
      DO 110 J=1,KP1
        J1=INDEX+J
  110   A(J)=ANUM(J1)/ADEN(K)
      ITST=KP1
      IWEVAL=1
      KOLD=K
C                                                                 C
C  IF THE STEPSIZE HAS BEEN CHANGED, REVISE Y TO CONFORM TO THE   C
C  NEW STEPSIZE.                                                  C
C                                                                 C
  120 IF (H.EQ.HOLD) GO TO 130
      RATIO=H/HOLD
      R1=1.0
      DO 125 J=2,KP1
        R1=R1*RATIO
        DO 125 I=1,NEQN
  125     Y(I,J)=SAVE(I,J)*R1
      ITST=KP1
      IWEVAL=1
C                                                                 C
C  SET LIMIT FOR TEST OF CONVERGENCE                              C
C                                                                 C
  130 BND=EPS/FLOAT(2*NEQN*(K+2))
C                                                                 C
C***********************                                          C
C                 ******                                          C
C  END BLOCK 1    ******                                          C
C                 ******                                          C
C*****************************************************************C
C                 ******                                          C
C  BEGIN BLOCK 2  ******                                          C
C                 ******                                          C
C***********************                                          C
C                                                                 C
C  SOLVE FOR Y AT T+H BY USING NEWTON ITERATION                   C
C                                                                 C
  200 T=T+H
C                                                                 C
C  OBTAIN STARTING VALUE FOR ITERATION BY MULTIPLYING Y BY        C
C  PASCAL TRIANGLE MATRIX, AND INITIALIZE ERROR(*) AND THE        C
C  ITERATION COUNTER ITER.                                        C
C                                                                 C
      DO 205 J=2,KP1
        DO 205 J1=J,KP1
```

```
          J2=K+J-J1
          DO 205 I=1,NEQN
  205       Y(I,J2)=Y(I,J2)+Y(I,J2+1)
      DO 210 I=1,NEQN
  210   ERROR(I)=0.0
      ITER=0
C                                                                 C
C  W(*,*) IS REEVALUATED ONLY IF CONVERGENCE HAS ALREADY FAILED, C
C  OR IF THERE HAS BEEN A CHANGE OF ORDER OR STEPSIZE.  W IS      C
C  NOT REEVALUATED DURING THE ITERATION.  (NEQN=1 IS TREATED      C
C  AS A SPECIAL CASE.)                                            C
C                                                                 C
      IF (IWEVAL.LT.1) GO TO 225
      CALL FCT(T,Y,YP)
      NFE=NFE+1
      CALL FDER(T,Y,W,YP,FCT)
      IWEVAL=-1
      IF (NEQN.EQ.1) GO TO 220
      AHINV=1./(A(1)*H)
      DO 215 I=1,NEQN
  215   W(I,I)=W(I,I)+AHINV
C                                                                 C
C  DECOMP PERFORMS AN LU FACTORIZATION OF W, WITH PARTIAL         C
C  PIVOTING.  IPS(*) IS USED TO HOLD INFORMATION ON THE ROW       C
C  INTERCHANGES.  C IS USED FOR WORKING STORAGE, AND J1 IS SET    C
C  TO -1 IF W IS FOUND TO BE SINGULAR.                            C
C                                                                 C
      CALL DECOMP(NEQN,W,IPS,C,J1)
      IF (J1.LT.0) GO TO 267
      GO TO 230
C                                                                 C
C  NEQN=1                                                         C
C                                                                 C
  220 W(1,1)=1.+W(1,1)*A(1)*H
      IF (W(1,1).EQ.0.0) GO TO 267
      GO TO 255
C                                                                 C
C  UP TO 3 ITERATIONS ARE TAKEN.  THE ITERATION IS STOPPED WHEN   C
C  THE NORM OF THE CORRECTION C(*) IS LESS THAN BND.  THE         C
C  SUCCESSIVE CORRECTIONS ARE ACCUMULATED IN ERROR(*).            C
C                                                                 C
  225 CALL FCT(T,Y,YP)
      NFE=NFE+1
      IF (NEQN.EQ.1) GO TO 255
C                                                                 C
C  SOLVE FOR C(*) USING THE FACTORIZED FORM OF W RETURNED BY      C
C  DECOMP.                                                        C
C                                                                 C
  230 IP=IPS(1)
      C(1)=(Y(IP,2)-YP(IP)*H)*AHINV
      DO 240 I=2,NEQN
        IP=IPS(I)
        IM1=I-1
        SUM=0.0
        DO 235 J=1,IM1
  235     SUM=SUM+W(IP,J)*C(J)
  240   C(I)=(Y(IP,2)-YP(IP)*H)*AHINV-SUM
      IP=IPS(NEQN)
      C(NEQN)=C(NEQN)/W(IP,NEQN)
      DO 250 IBACK=2,NEQN
        I=NEQN-IBACK+1
        IP=IPS(I)
        IP1=I+1
        SUM=0.0
        DO 245 J=IP1,NEQN
  245     SUM=SUM+W(IP,J)*C(J)
  250   C(I)=(C(I)-SUM)/W(IP,I)
      GO TO 260
C                                                                C
C  NEQN=1                                                        C
C                                                                C
  255 C(1)=(Y(1,2)-YP(1)*H)/W(1,1)
C                                                                C
C  APPLY THE CORRECTION TO Y(*,1) AND Y(*,2), AND ADD IT TO      C
C  ERROR(*).  COMPUTE THE NORM OF C AND TEST WHETHER             C
C  CONVERGENCE HAS BEEN ACHIEVED.                                C
C                                                                C
  260 CNORM=0.
      DO 265 I=1,NEQN
        Y(I,1)=Y(I,1)+A(1)*C(I)
        Y(I,2)=Y(I,2)-C(I)
        ERROR(I)=ERROR(I)+C(I)
  265   CNORM=AMAX1(CNORM,ABS(C(I)/WT(I)))
      IF (CNORM.LE.BND) GO TO 285
C                                                                C
C  NO CONVERGENCE YET.  PERFORM ANOTHER ITERATION UNLESS THIS    C
C  WAS THE THIRD.                                                C
C                                                                C
      ITER=ITER+1
      IF (ITER.LT.3) GO TO 225
C                                                                C
C  THE ITERATION DID NOT CONVERGE.  W WILL BE REEVALUATED FOR    C
C  ANOTHER ATTEMPT.  H IS REDUCED TO H/4., UNLESS THIS IS THE    C
C  FIRST FAILURE AND W WAS NOT REEVALUATED FOR IT.               C
C                                                                C
  267 T=TOLD
      IF (IWEVAL.EQ.0) GO TO 275
      H=0.25*H
      IF (ABS(H).LT.HMIN) GO TO 275
C                                                                C
C  RESTORE Y(*,1) AND GO TO BLOCK 1 FOR AN ATTEMPT WITH THE NEW  C
C  STEPSIZE.                                                     C
C                                                                C
      DO 270 I=1,NEQN
  270   Y(I,1)=SAVE(I,1)
      GO TO 120
C                                                                C
C  RESTORE Y                                                     C
C                                                                C
  275 DO 280 J=1,KP1
        DO 280 I=1,NEQN
  280     Y(I,J)=SAVE(I,J)
C                                                                C
C  IF THIS IS THE FIRST CONVERGENCE FAILURE, TRY AGAIN WITH W    C
C  REEVALUATED.                                                  C
C                                                                C
      IWEVAL=1
      IF (ABS(H).GE.HMIN) GO TO 200
C                                                                C
C  CONVERGENCE COULD NOT BE ACHIEVED AT THE SMALLEST POSSIBLE    C
C  STEPSIZE.  INCREASE EPS TO A VALUE WHICH WILL ALLOW           C
C  CONVERGENCE AND RETURN TO CALLING PROGRAM.                    C
C                                                                C
      GFACT=CNORM/BND
      IF (GFACT.LT.2.) GFACT=2.
      EPS=EPS*GFACT*(1.+FOURU)
      H=SIGN(HMIN,H)
      ICRASH=3
      RETURN
C                                                                C
C                                                                C
C  CONVERGENCE WAS ACHIEVED.  TEST WHETHER THE SOLUTION MEETS    C
C  THE ERROR TOLERANCE.                                          C
C                                                                C
  285 IWEVAL=0
      D=0.
      DO 290 I=1,NEQN
  290   D=AMAX1(D,ABS(ERROR(I)/WT(I)))
      D=D/(ERTST(K)*EPS)
      IF (D.LE.1.) GO TO 400
C                                                                C
C************************                                        C
C                 ******                                         C
C  END BLOCK 2    ******                                         C
C                 ******                                         C
C****************************************************************C
C                 ******                                         C
C  BEGIN BLOCK 3  ******                                         C
C                 ******                                         C
C************************                                        C
C                                                                C
C  THE STEP WAS NOT SUCCESSFUL.  THE STEPSIZE IS REDUCED FOR THE C
C  NEXT STEP.  IF K.GE.2, THE REDUCTION OF THE ORDER IS ALSO     C
C  CONSIDERED.                                                   C
C                                                                C
C  RESTORE T AND Y(*,1)                                          C
```

```
C                                                                C
      T=TOLD
      DO 310 I=1,NEQN
  310   Y(I,1)=SAVE(I,1)
C                                                                C
C  CHECK POSSIBILITIES FOR CONTINUATION.                         C
C                                                                C
      IFAIL=IFAIL+1
      IF (IFAIL.GT.3) GO TO 320
      IF ((IFAIL.LT.3).AND.(K.GT.1)) GO TO 325
C                                                                C
C  THIS IS THE THIRD FAILURE, OR ELSE K=1 ON THE FIRST OR SECOND C
C  FAILURE.  H IS HALVED, Y(*,2) IS EVALUATED, AND THE ORDER IS  C
C  SET TO 1 FOR THE NEXT ATTEMPT.                                C
C                                                                C
      IFAIL=3
      H=0.5*H
      IF (ABS(H).LT.HMIN) H=SIGN(HMIN,H)
      CALL FCT(T,Y,YP)
      NFE=NFE+1
      DO 315 I=1,NEQN
        SAVE(I,2)=YP(I)*H
  315   Y(I,2)=SAVE(I,2)
      HOLD=H
      K=1
      KP1=2
      A(1)=-1.
      KOLD=1
      ITST=2
      IWEVAL=1
      GO TO 200
C                                                                C
C  IFAIL.GT.3 AND K=1.  CHOOSE OPTIMUM STEPSIZE FOR NEXT ATTEMPT.C
C                                                                C
  320 H=H/SQRT(2.*D)
      IF (ABS(H).LT.HMIN) GO TO 340
      GO TO 120
C                                                                C
C  FIRST OR SECOND FAILURE AND K.GE.2.  TEST WHETHER ORDER       C
C  SHOULD BE LOWERED, AND CHOOSE STEPSIZE FOR NEXT ATTEMPT.      C
C                                                                C
  325 R=1.2*(2.*D)**(1./FLOAT(KP1))
      D1=0.
      DO 330 I=1,NEQN
  330   D1=AMAX1(D1,ABS(Y(I,KP1)/WT(I)))
      RDWN=1.3*(2.*D1/(ERDWN(K)*EPS))**(1./FLOAT(K))
      IF (RDWN.LT.R) GO TO 335
C                                                                C
C  THE ORDER IS NOT CHANGED.  THE STEP DECREASE IS LIMITED TO A  C
C  FACTOR OF 1/10.                                               C
C                                                                C
      IF (R.GT.10.) R=10.
      H=H/R
      IF (ABS(H).LT.HMIN) GO TO 340
      GO TO 120
C                                                                C
C  THE ORDER IS LOWERED.  THE STEP IS DECREASED BY AT LEAST 9    C
C  PERCENT, BUT THE DECREASE IS LIMITED TO A FACTOR OF 1/10.     C
C                                                                C
  335 K=K-1
      IF (RDWN.LT.1.1) RDWN=1.1
      IF (RDWN.GT.10.) RDWN=10.
      H=H/R
      IF (ABS(H).GE.HMIN) GO TO 100
      KP1=K+1
C                                                                C
C  ERROR TOLERANCE CANNOT BE MET AT SMALLEST ALLOWED STEPSIZE.   C
C  INCREASE TOLERANCE, RESTORE Y, AND RETURN TO CALLING PROGRAM. C
C                                                                C
  340 GFACT=(HMIN/ABS(H))**KP1
      IF (GFACT.LT.2.) GFACT=2.
      EPS=EPS*GFACT*(1.+FOURU)
      H=SIGN(HMIN,H)
      DO 345 J=2,KP1
        DO 345 I=1,NEQN
  345     Y(I,J)=SAVE(I,J)
      ICRASH=4
      RETURN
C                                                                C
C***********************                                         C
C                 ******                                         C
C  END BLOCK 3    ******                                         C
C                 ******                                         C
C****************************************************************C
C                 ******                                         C
C  BEGIN BLOCK 4  ******                                         C
C                 ******                                         C
C***********************                                         C
C                                                                C
C  THE STEP WAS SUCCESSFUL.  STORE THE STEPSIZE USED AND CORRECT C
C  THE REMAINING COMPONENTS OF Y.                                C
C                                                                C
  400 HOLD=H
      IF (KP1.LT.3) GO TO 410
      DO 405 J=3,KP1
        DO 405 I=1,NEQN
  405     Y(I,J)=Y(I,J)+A(J)*ERROR(I)
C                                                                C
  410 ITST=ITST-1
C                                                                C
C  REVISION OF ORDER AND STEPSIZE IS CONSIDERED ONLY IF ITST=0.  C
C                                                                C
      IF (ITST.EQ.0) GO TO 420
      IF ((ITST.GT.1).OR.(K.EQ.6)) RETURN
```

```
C                                                                       C
C  ITST=1 AND K.NE.6 -- SAVE ERROR(*) IN Y(*,7).  IF NEXT STEP          C
C  SUCCEEDS, IT WILL BE USED IN DECIDING WHETHER TO RAISE ORDER.        C
C                                                                       C
      DO 415 I=1,NEQN
  415   Y(I,7)=ERROR(I)
      RETURN
C                                                                       C
C  DETERMINE APPROPRIATE ORDER AND STEPSIZE FOR NEXT STEP.              C
C                                                                       C
  420 KD=0
      R=1.2*(2.*D)**(1./FLOAT(KP1))
C                                                                       C
C  IF K=1, CANNOT LOWER ORDER. OTHERWISE, CHECK WHETHER ORDER           C
C  K-1 ALLOWS LARGER STEPSIZE.                                          C
C                                                                       C
      IF (K.EQ.1) GO TO 430
      D1=0.
      DO 425 I=1,NEQN
  425   D1=AMAX1(D1,ABS(Y(I,KP1)/WT(I)))
      RDWN=1.3*(2.*D1/(ERDWN(K)*EPS))**(1./FLOAT(K))
      IF (RDWN.GT.R) GO TO 430
C                                                                       C
C  ORDER K-1 IS BETTER THAN ORDER K.                                    C
C                                                                       C
      R=RDWN
      KD=-1
C                                                                       C
C  IF K=6, CANNOT RAISE ORDER.  OTHERWISE, CHECK WHETHER ORDER          C
C  K+1 ALLOWS LARGER STEPSIZE.                                          C
C                                                                       C
C                                                                       C
  430 IF (K.EQ.6) GO TO 445
      D1=0.
      DO 435 I=1,NEQN
  435   D1=AMAX1(D1,ABS((ERROR(I)-Y(I,7))/WT(I)))
      RUP=1.4*(2.*D1/(ERUP(K)*EPS))**(1./FLOAT(K+2))
      IF (RUP.GT.R) GO TO 445
C                                                                       C
C  ORDER WILL BE RAISED, SO APPEND A NEW COLUMN TO Y.                   C
C                                                                       C
      KD=1
      R=RUP
      D1=A(KP1)/FLOAT(KP1)
      DO 440 I=1,NEQN
  440   Y(I,KP1+1)=ERROR(I)*D1
C                                                                       C
C  IF ORDER IS UNCHANGED AND THE STEPSIZE CHANGE WOULD BE LESS          C
C  THAN 12 PERCENT INCREASE, OR LESS THAN 1 PERCENT DECREASE, NO        C
C  CHANGE IS MADE.  OTHERWISE, THE NEW ORDER AND STEPSIZE ARE           C
C  SET.  ANOTHER TEST WILL BE MADE AFTER K+1 STEPS.  STEP               C
C  INCREASE IS LIMITED TO A FACTOR OF 2.                                C
```

```
C                                                                      C       78900
  445 ITST=KP1                                                                 79000
      IF ((KD.EQ.0).AND.(ABS(R-0.95).LT.0.06)) RETURN                          79100
      K=K+KD                                                                   79200
      IF (R.LT.0.5) R=0.5                                                      79300
      H=SIGN(AMAX1(ABS(H/R),FOURU*ABS(T)),H)                                   79400
      RETURN                                                                   79500
C                                                                      C       79600
C************************                                              C       79700
C                  ******                                              C       79800
C  END BLOCK 4     ******                                              C       79900
C                  ******                                              C       80000
C**********************************************************************C       80100
C                                                                      C       80200
C  END OF GEAR                                                         C       80300
C                                                                      C       80400
      END                                                                      80500


      SUBROUTINE DECOMP(N,A,IPS,SCALES,JS)                                     10000
C                                                                      C       10100
C**********************************************************************C       10200
C                                                                      C       10300
C  THE NUMERICAL SOLUTION OF THE SYSTEM OF LINEAR EQUATIONS            C       10400
C  AX = B IS VERY SIMPLE ONCE THE N BY N MATRIX A HAS BEEN             C       10500
C  FACTORED INTO A PRODUCT OF LOWER AND UPPER TRIANGULAR               C       10600
C  MATRICES.                                                           C       10700
C                                                                      C       10800
C  DECOMP USES GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING AND          C       10900
C  SCALING TO FIND A TRANSPOSITION MATRIX P AND LOWER AND UPPER        C       11000
C  TRIANGULAR MATRICES L AND U SUCH THAT ALL THE DIAGONAL              C       11100
C  ELEMENTS OF L ARE 1 AND PLU = A.                                    C       11200
C                                                                      C       11300
C  L AND U ARE STORED IN PLACE OF A AS THE MATRIX P(L-I)+PU            C       11400
C  = P(L-I+U)                                                          C       11500
C                                                                      C       11600
C  THE PERMUTATION MATRIX IS SPECIFIED BY IPS(*).  ALL THE             C       11700
C  ELEMENTS OF THE K-TH COLUMN OF P ARE ZERO, EXCEPT FOR THE           C       11800
C  IPS(K)-TH, WHICH IS 1.  (NOTE THAT THE TRANSPOSE OF P IS            C       11900
C  ITS INVERSE.)                                                       C       12000
C                                                                      C       12100
C  SCALES(*) IS USED FOR WORKING STORAGE.                              C       12200
C                                                                      C       12300
C  DECOMP RETURNS WITH JS=1 IF THE FACTORIZATION IS SUCCESSFUL,        C       12400
C  OR WITH JS=-1 IF A IS FOUND TO BE SINGULAR.                         C       12500
C                                                                      C       12600
C**********************************************************************C       12700
C                                                                      C       12800
C  THIS SUBROUTINE IS WRITTEN IN SINGLE PRECISION.                     C       12900
C  TO OBTAIN A DOUBLE PRECISION VERSION, REMOVE THE C FROM             C       13000
C  COLUMN 1 OF THE NEXT FOUR CARDS.                                    C       13100
C     DOUBLE PRECISION A,ABS,AMAX1,BIG,DABS,DMAX1,D1,D2,PIVOT,                 13200
```

```
C    1                  ROWNRM,SCALES,SIZE
C     ABS(D1)=DABS(D1)
C     AMAX1(D1,D2)=DMAX1(D1,D2)
C                                                                      C
      DIMENSION A(N,N),IPS(N),SCALES(N)
C                                                                      C
      JS=-1
C                                                                      C
      DO 200 I=1,N
         IPS(I)=I
         ROWNRM=0.0
         DO 100 J=1,N
  100       ROWNRM=AMAX1(ROWNRM,ABS(A(I,J)))
         IF(ROWNRM.EQ.0.0) RETURN
  200    SCALES(I)=1.0/ROWNRM
C                                                                      C
      NM1=N-1
      DO 400 I=1,NM1
         BIG=0.0
         DO 300 J=1,N
            JP=IPS(J)
            SIZE=ABS(A(JP,I))*SCALES(JP)
            IF(SIZE.LE.BIG) GO TO 300
            BIG=SIZE
            IDXPIV=J
  300       CONTINUE
         IF(BIG.EQ.0.0) RETURN
         IF(IDXPIV.EQ.I) GO TO 350
         K=IPS(I)
         IPS(I)=IPS(IDXPIV)
         IPS(IDXPIV)=K
  350    IP=IPS(I)
         PIVOT=A(IP,I)
         IP1=I+1
         DO 400 J=IP1,N
            JP=IPS(J)
            A(JP,I)=A(JP,I)/PIVOT
            DO 400 K=IP1,N
  400          A(JP,K)=A(JP,K)-A(JP,I)*A(IP,K)
C                                                                      C
      KP=IPS(N)
      IF(A(KP,N).EQ.0.0) RETURN
C                                                                      C
      JS=1
C                                                                      C
      RETURN
C                                                                      C
      END


      SUBROUTINE GINTRP(TOUT,YOUT,YPOUT,Y)
C                                                                      C
```

```
C****************************************************************C
C                                                                C
C  SUBROUTINE GINTRP MAY BE USED IN CONJUNCTION WITH SUBROUTINE  C
C  GEAR TO INTERPOLATE THE SOLUTION TO THE NONMESH POINT TOUT.   C
C  GINTRP SHOULD BE CALLED WHEN THE INTEGRATION HAS PROCEEDED TO C
C  THE FIRST MESH POINT BEYOND TOUT.                             C
C                                                                C
C  SUBROUTINE GEAR APPROXIMATES THE SOLUTION FOR X NEAR T BY A   C
C  K DEGREE POLYNOMIAL P(X).  THE COEFFICIENT OF THE S**L TERM   C
C  OF P(X) IS STORED IN Y(*,L+1), L=0,1,...,K, WHERE             C
C  S = (X-T)/HOLD                                                C
C                                                                C
C  SUBROUTINE GINTRP APPROXIMATES THE SOLUTION AND ITS           C
C  DERIVATIVE AT TOUT BY YOUT(*)=P(TOUT) AND YPOUT(*)=P'(TOUT),  C
C  RESPECTIVELY.                                                 C
C                                                                C
C  THE PARAMETERS IN THE ARGUMENT LIST ARE                       C
C                                                                C
C    TOUT -- THE VALUE OF THE INDEPENDENT VARIABLE FOR WHICH     C
C            THE SOLUTION IS TO BE FOUND.                        C
C    YOUT -- AN ARRAY OF DIMENSION NEQN TO HOLD THE VALUE OF     C
C            THE SOLUTION AT TOUT                                C
C    YPOUT -- AN ARRAY OF DIMENSION NEQN TO HOLD THE VALUE OF    C
C             THE DERIVATIVE AT TOUT.                            C
C    Y -- THE ARRAY OF DIMENSIONS (NEQN,7) RETURNED AT T BY      C
C          SUBROUTINE GEAR.                                      C
C                                                                C
C  THE VALUES OF T, HOLD, NEQN, AND K ARE PASSED FROM SUBROUTINE C
C  GEAR THROUGH THE COMMON BLOCK GEDATA.                         C
C                                                                C
C****************************************************************C
C                                                                C
C  THIS PROGRAM IS WRITTEN IN SINGLE PRECISION.  TO OBTAIN A     C
C  DOUBLE PRECISION VERSION, REMOVE THE C FROM COLUMN 1 OF THE   C
C  NEXT TWO CARDS.                                               C
C     DOUBLE PRECISION T,HOLD,RVAR,TOUT,YOUT,YPOUT,Y,S,FACT,
C    1                 TEMP1,TEMP2
C                                                                C
      LOGICAL LVAR
C                                                                C
      DIMENSION YOUT(NEQN),YPOUT(NEQN),Y(NEQN,7),FACT(7)
C                                                                C
      COMMON /GEDATA/ T,HOLD,RVAR(10),NEQN,K,IVAR(5),LVAR
C                                                                C
      S=(TOUT-T)/HOLD
      KP1=K+1
      KP2=K+2
C                                                                C
      DO 100 J=2,KP1
  100   FACT(J)=FLOAT(J-1)/HOLD
C                                                                C
      DO 300 I=1,NEQN
```

```
        TEMP1=0.0
        TEMP2=0.0
        DO 200 JBACK=1,K
          J=KP2-JBACK
          TEMP1=TEMP1*S+Y(I,J)
  200     TEMP2=TEMP2*S+Y(I,J)*FACT(J)
        YOUT(I)=TEMP1*S+Y(I,1)
  300   YPOUT(I)=TEMP2
C                                                                C
      RETURN
C                                                                C
C  END OF GINTRP                                                 C
C                                                                C
      END


      SUBROUTINE QJACOB(T,Y,W,YP,FCT)
C                                                                C
C****************************************************************C
C                                                                C
C  THIS SUBROUTINE MAY BE USED WITH GEAR TO CALCULATE THE        C
C  JACOBIAN MATRIX OF F(T,Y) BY NUMERICAL DIFFERENCING.         C
C  NORMALLY, QJACOB SHOULD BE USED ONLY IF THE JACOBIAN IS SO   C
C  COMPLEX AS TO MAKE A DIRECT CALCULATION INCONVENIENT.        C
C                                                                C
C****************************************************************C
C                                                                C
C  THIS PROGRAM IS WRITTEN IN SINGLE PRECISION.                  C
C  TO OBTAIN A DOUBLE PRECISION VERSION, REMOVE THE C FROM       C
C  COLUMN 1 OF THE NEXT FOUR CARDS.                              C
C     DOUBLE PRECISION T,Y,W,YP,RV1,RV2,EPS,D1,D2,ABS,DABS,
C    1                 AMAX1,DMAX1,DEL,YY,FOURU
C     ABS(D1)=DABS(D1)
C     AMAX1(D1,D2)=DMAX1(D1,D2)
C                                                                C
      LOGICAL LV
C                                                                C
      COMMON /GEDATA/ RV1(3),EPS,RV2(8),NEQN,K,NFE,IV(4),LV
C                                                                C
      DIMENSION Y(NEQN),YP(NEQN),W(NEQN,NEQN)
C                                                                C
C  IN THE FOLLOWING DATA STATEMENT, FOURU MUST BE SET TO A       C
C  VALUE APPROPRIATE TO THE MACHINE BEING USED.                  C
C                                                                C
C  FOURU IS 4.*U, WHERE THE COMPUTER UNIT ROUNDOFF ERROR U IS    C
C  THE SMALLEST POSITIVE VALUE REPRESENTABLE IN THE MACHINE      C
C  SUCH THAT (1.+U).GT.1.                                        C
C                                                                C
C  SOME REPRESENTATIVE VALUES ARE                                C
C    U = 9.54E-7      FOURU = 3.82E-6       IBM 360/370 S.P.     C
C    U = 5.96E-8      FOURU = 2.39E-7       PDP-11 S.P.          C
C    U = 1.49E-8      FOURU = 5.97E-8       UNIVAC 1108          C
```

```
C    U = 7.45E-9       FOURU = 2.99E-8       PDP-10                    C
C    U = 7.11E-15      FOURU = 2.85E-14      CDC 6000 SERIES           C
C    U = 2.22D-16      FOURU = 8.89D-16      IBM 360/370 D.P.          C
C    U = 1.39D-17      FOURU = 5.56D-17      PDP-11 D.P.               C
C                                                                      C
      DATA FOURU/2.39E-7/
C                                                                      C
C                                                                      C
      DO 200 I=1,NEQN
        YY=Y(I)
        DEL=AMAX1(EPS**2,AMAX1(EPS,FOURU)*ABS(YY))
        Y(I)=Y(I)+DEL
        CALL FCT(T,Y,W(1,I))
        DO 100 J=1,NEQN
 100      W(J,I)=(W(J,I)-YP(J))/DEL
 200    Y(I)=YY
C                                                                      C
      NFE=NFE+NEQN
C                                                                      C
      RETURN
C                                                                      C
      END
```

APPENDIX D

SOLVING PARTIAL DIFFERENTIAL EQUATIONS

Models describing the transport of substances through continuous media are naturally represented as initial value problems for systems of partial differential equations. A simple example illustrating the essential features of such a model is

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} - V\frac{\partial u}{\partial x} - au + bv \qquad \text{(a)}$$

$$\frac{\partial v}{\partial t} = au - bv \qquad \text{(b)} \qquad \text{(D-1)}$$

$$u(x,t_0) = f(x) \qquad v(x,t_0) = g(x) \qquad \text{(c)}$$

Here, $u(x,t)$ may be regarded as the concentration of a chemical species dissolved in the water of a (one dimensional) stream which flows with velocity V. $v(x,t)$ is the concentration of the same species bound to the sediments of the stream bed. The first term on the right hand side of (a) describes diffusion through the water, the second term accounts for the motion of the water, and the remaining terms model the exchange of the species between water and sediment. A more complex and realistic model of this sort is the mercury model described in Lassiter(1976).

A general approach to solving (D-1) is to approximate it by a system of first order ordinary differential equations. This is done by a process of *semidiscretization*:

(1) The spatial dependence of $u(x,t)$ and $v(x,t)$ is represented by $u_i(t)$ and $v_i(t)$, which approximate $u(x_i,t)$ and $v(x_i,t)$ on a set of grid points x_i, i=1,2,...,N.

(2) The partial derivatives appearing in (a) are approximated by finite differences of the u_i's.

The resulting approximation is

$$u_i'(t) = D\Delta^2(i) - V\Delta^1(i) - au_i + bv_i$$

$$v_i'(t) = au_i - bv_i \qquad \text{(D-2)}$$

$$u_i(t_0) = f(x_i) \qquad v_i(t_0) = g(x_i) \qquad i = 1, 2, \ldots, N$$

where $\Delta^1(i)$ and $\Delta^2(i)$ represent the finite difference approximations of $\partial u/\partial x$ and $\partial^2 u/\partial x^2$ at (x_i,t). (D-2) is a system of 2N coupled first order ordinary differential equations which may be solved numerically using the codes in this report, once the forms of Δ^1 and Δ^2 are specified. To keep considerations simple, suppose that the grid points are chosen to be equally spaced:

$$x_i = x_0 + ih$$

Then several possibilities for the finite differences immediately come to mind,

for $\partial u/\partial x$:

$$\Delta^1(i) = \Delta_1^1(i) = \frac{u_{i+1} - u_i}{h} \qquad \text{(forward difference)}$$

$$\Delta^1(i) = \Delta_2^1(i) = \frac{u_i - u_{i-1}}{h} \qquad \text{(backward difference)}$$

$$\Delta^1(i) = \Delta_3^1(i) = \frac{u_{i+1} - u_{i-1}}{2h} \qquad \text{(central difference)}$$

for $\partial^2 u/\partial x^2$:

$$\Delta^2(i) = \Delta_1^2(i) = \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2}$$

$$\Delta^2(i) = \Delta_2^2(i) = \frac{\Delta_3^1(i+1) - \Delta_3^1(i-1)}{2h} = \frac{u_{i+2} - 2u_i + u_{i-2}}{4h^2}$$

Considerable care is necessary in choosing which approximation to use. A particularly bad choice is to use Δ_2^2 for $\partial^2 u/\partial x^2$. The unsuitability of this approximation can best be appreciated by considering the special case $V=0$. Then the even- and odd-numbered grid points are completely uncoupled, so that the problem being solved is actually two independent problems, each of which uses only half of the available information. Since any errors occurring in the even (odd) points are propagated only among the even (odd) points, a common manifestation in the solution is the appearance of an unphysical oscillatory behavior in the spatial variable. (Such oscillations are visible in the computed solutions shown in the reference cited above.) A similar, though less easily visualizable, difficulty occurs with the use of Δ_3^1 for $\partial u/\partial x$, because $\Delta_3^1(i)$ is independent of the value of u_i. In general, it is safest to use Δ_1^2 in combination with either Δ_1^1 or Δ_2^1, so that (D-1) might be approximated by either

$$\begin{aligned} u_i' &= \frac{D}{h^2}(u_{i+1}-2u_i+u_{i-1}) - \frac{V}{h}(u_{i+1}-u_i) - au_i + bv_i \\ v_i' &= au_i - bv_i \qquad\qquad \text{(D-3a)} \\ u_i(t_0) &= f(x_i) \qquad v_i(t_0) = g(x_i) \qquad i = 1, 2, \ldots, N \end{aligned}$$

or

$$\begin{aligned} u_i' &= \frac{D}{h^2}(u_{i+1}-2u_i+u_{i-1}) - \frac{V}{h}(u_i-u_{i-1}) - au_i + bv_i \\ v_i' &= au_i - bv_i \qquad\qquad\qquad\qquad\qquad\qquad \text{(D-3b)} \\ u_i(t_0) &= f(x_i) \qquad v_i(t_0) = g(x_i) \qquad i = 1, 2, \ldots, N \end{aligned}$$

It is important to understand that the problem presented to the integration routines is (D-3), which is only an approximation of (D-1). While requesting increasingly accurate error tolerances will produce increasingly accurate solutions of (D-3), the integration routines cannot reduce the errors incurred in going from (D-1) to (D-3). Unless care is taken with this initial approximation, it may be found that the solution of (D-3) has properties quite different from those expected for the solution of (D-1).

Besides the choice of Δ^1 and Δ^2, the most important factor affecting the accuracy of the semidiscretization process is the spacing h of the spatial grid. If $u(x,t)$ or $v(x,t)$ varies rapidly on a scale smaller than h, then the finite difference approximations of $\partial u/\partial x$ and $\partial^2 u/\partial x^2$ cannot be accurate, no matter what form is chosen. In particular, h must be small enough that the initial conditions $f(x)$ and $g(x)$ are accurately represented. The quest for increased accuracy can quickly lead to very large systems of equations, since the number of equations doubles each time h is halved. Computing time and storage requirements will often limit the accuracy which it is feasible to achieve.

Additionally, the grid must be finite in extent. In effect, one cannot approximate the initial value problem (D-1), but only the initial-boundary value problem which is (D-1) with the additional condition that $u(x_1,t) = v(x_1,t) = u(x_N,t) = v(x_N,t) = 0$. Thus, the approximation cannot be accurate unless $u(x,t)$ and $v(x,t)$ are of negligible magnitude near x_1 and x_N over the entire interval $[t_0,T]$ on which the integration is performed.

Given the model (D-3a) or (D-3b), it is tempting to use it to follow the fate of a "point spill" of pollutant at x_k by using the initial conditions

$$\begin{aligned} &u_k(t_0) = A, \qquad u_i(t_0) = 0 \text{ for } i \neq k \\ &v_i(t_0) = 0 \text{ for all } i \end{aligned} \qquad \text{(D-4)}$$

If this is done for (D-3a), one immediately finds at t_0,

$$\begin{aligned} u_{k-1}' &= \frac{D}{h^2}A - \frac{V}{h}A \\ u_k' &= -2\frac{D}{h^2}A + \frac{V}{h}A - aA \\ u_{k+1}' &= \frac{D}{h^2}A \end{aligned}$$

If V is sufficiently large, u'_{k-1} will be negative, so that u_{k-1} will immediately assume a (physically meaningless) negative value. (The situation would be even worse if the model had incorporated the differences Δ_3^1 and Δ_2^2, which will always produce a negative value for u_{k-1}, regardless of the value of V.) For (D-3b), however, the derivatives at t_0 are

$$u'_{k-1} = \frac{D}{h^2}A$$

$$u'_{k} = -2\frac{D}{h^2}A - \frac{V}{h}A - aA$$

$$u'_{k+1} = \frac{D}{h^2}A + \frac{V}{h}A$$

and the problem of negative values does not occur.

The real difficulty here is that the grid spacing h is not sufficiently small to accurately represent the initial condition. Indeed, the smooth function $f(x) = u(x,t_0)$ which (D-4) is supposed to approximate has not even been specified. Nevertheless, it may be permissible to ignore the origins of (D-3b), and to use it and (D-4) as a sufficiently accurate model of the system being studied. If this is done, it must be borne in mind that the grid spacing h becomes an essential feature of the model: changing h will produce a quantitatively different solution.

This discussion has been intended only to illuminate some of the pitfalls which may be encountered in solving problems similar to (D-1). A thorough exploration of the numerical solution of partial differential equations may be found in the text of Ames(1969).